U.PORTO

MESTRADO
MULTIMÉDIA - ESPECIALIZAÇÃO EM MÚSICA INTERACTIVA E DESIGN DE SOM

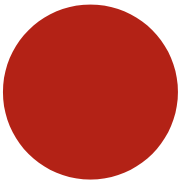
Procedural Generation of Musical Metrics Based on Lyrics Analysis

Urbano Ferreira

M
2016

FACULDADES PARTICIPANTES:

FACULDADE DE ENGENHARIA
FACULDADE DE BELAS ARTES
FACULDADE DE CIÊNCIAS
FACULDADE DE ECONOMIA
FACULDADE DE LETRAS



U.PORTO

Procedural Generation of Musical Metrics Based on Lyrics Analysis

Urbano Ferreira

Mestrado em Multimédia da Universidade do Porto

Orientador: Rui Penha (Doutor)

Coorientador: George Sioros (Doutor)

Junho de 2016

Procedural Generation of Musical Metrics Based on Lyrics Analysis

Urbano Ferreira

Mestrado em Multimédia da Universidade do Porto

Aprovado em provas públicas pelo Júri:

Presidente: Professor Doutor Rui Rodrigues

Vogal Externo: Professor Doutor Dimitris Andrikopoulos

Orientador: Professor Doutor Rui Penha

Resumo

Mais do que a componente semântica e discursiva, as letras musicais contêm geralmente outro tipo de informação, que mais do que com o ato da escrita, tem que ver com o ato da pronúncia. Assumindo que uma letra musical é escrita para posteriormente ser reproduzida, há um cuidado para que esse processo nos transmita algo também, completamente diferente daquilo que nos é transmitido pela letra no papel. A sincronia das acentuações fonéticas e lexicais da letra com as componentes musicais em que se insere é disso o maior exemplo. A ideia nasce da vontade de criar um sistema capaz de devolver informação musical para uma dada letra. Neste projeto, proponho-me a dar um primeiro passo sólido para um futuro desenvolvimento dessa ideia, devolvendo uma estrutura métrica unicamente baseada na análise da letra. Para o efeito, utilizarei o CMUdict, um dicionário de informação fonética para a língua inglesa que contém, para cada palavra, a divisão por fonemas com os respectivos marcadores referentes à sua acentuação lexical. Todo o funcionamento do sistema será baseado na linguagem de programação Python, tendo sido todo o código desenvolvido para o projeto. Para cada letra introduzida, é executada uma análise por versos e encontrada a estrutura métrica que melhor se adapta a todos eles.

Abstract

More than semantic and discursive components, lyrics often contain other kind of information. This information has to do not with the act of writing, but with the act of pronunciation. Assuming that lyrics are written to later be reproduced, there is a care for this process to pass us something too, completely different from what is conveyed by the lyrics on paper. The synchrony of phonetic and lexical accents of the lyrics with musical events is a great example of that. The idea rises from the will to create a system able to return music information for a given lyric. In this project, the purpose was to take a solid first step for a future development of this idea. To this end, it was used the CMUdict, a phonetic information dictionary for English language that contains, for each word, the division of its phonemes with the respective markers related to their stress. The entire operation of the system is based on Python programming language, having all the code been developed for the project. For each lyric, it analyses all the verses and finds an appropriate metric structure.

Agradecimentos

Ao Professor Doutor Rui Penha,

pela orientação e disponibilidade apresentadas.

Ao Doutor George Sioros,

pelo interesse e sabedoria que demonstrou.

Aos meus pais,

por serem a base que me suporta.

Aos meus amigos.

Index

1 Introduction 1
1.1 Motivation 1
1.2 Description of the Work 1
1.3 Why Lullabies? 2
1.4 The research 2
1.5 Structure of the Dissertation 2

2 State of the Art 4
2.1 Introduction 4
2.2 Musical Rhythm and Meter 4
2.3 Prosody 5
2.4 Related Work 5

3 From the Lyrics Input to the Metric Output 7
3.1 Introduction 7
3.2 Lyrics Analysis 7
3.2.1 Lexical Stress extraction 8
3.3 Mapping of Prosodic Accents to the Metric Grid 9
3.3.1 Generation of Metrical Templates for Prosody 10
3.3.2 Recursive Alignment of Verses in the Templates 11
3.3.3 Selection of the Best Template 14
3.4 Results Analysis and Output 17

4 Implementation 19
4.1 CMUdict 19
4.2 Natural Language Toolkit (NLTK) 20
4.3 Python 21
4.3.1 Brief Explanation of Python Code 21

5 Application to six lyrics/poems 23
5.1 Lullabies 23
5.1.1 All the Pretty Horses 23
5.1.2 Hush, Little Baby 26
5.2 Children's Poetry 30
5.2.1 All Things Bright and Beautiful 30
5.2.2 Bed in Summer 34
5.2.3 The Fieldmouse 38
5.2.4 The Lamb 43

6 Conclusion 47
6.1 Summary 48
6.2 Contribution of the Work 49
6.3 Future Work 49
6.3.1 The Analysis 49
6.3.2 The Output 50
6.3.3 User Interaction 51
7 Bibliography 53
8 Appendix 55
8.1 Python Code 55

1 Introduction

1.1 Motivation

The access to music creation tools has been democratized in the last decades, increasing the opportunities of amateur musicians to create and explore in the music universe. It was in this line of thinking that this project was first thought.

The process of musical creation is multidisciplinary and complex. And it doesn't stick to the basic musical dimensions. In most of the cases, it explores fields as psychology and linguistics and this can be challenging, mainly when you're creating music alone and you don't have the needed expertise in some of these disciplines.

Due to this multidisciplinary approach, the music creation process can be developed in several ways. There are musicians who choose to first create the musical arrangement and then from it, generate an appropriate lyric. Other first set a mood and create through it the lyrics and arrangement. There is another method that is widely used, which is to create a musical accompaniment to a particular lyric.

1.2 Description of the Work

The main idea behind this thesis project is to create a tool to help those who start the musical creation process by/from the lyrics. Assuming that the lyrics and the musical arrangement are directly related, I created a system able to return a musical metric distribution for a given lyric. The project was designed as a starting point to something bigger, such as a complete system of musical generation from lyrics. And, independently of what comes next, it was with this idea as motivation that I faced this challenge.

The prototype was developed to analyze lyrics (in English) that respect some basic format rules and extract some prosodic information from it, based on a specific toolkit for language processing. After the text file is parsed, it stops working with words and starts working with numbers. These numbers (markers) represent the lexical stress levels of the words and are organized in sequences, by verses. Each one of these sequences then

becomes a metric template. Based on the operations that are explained in the chapter 3, one of this templates is chosen as the best and all the verses are distributed in the metric grid according to it.

An important and time-consuming part was the study of programming language, Python. This step has been set in the initial schedule and allowed a greater workflow later.

1.3 Why Lullabies?

The lyrics used as test while developing the system were lullabies, because of their simple and constant structure. From verse to verse, the length and number of lexical stress does not change that much and generally they don't have a chorus. And even if they do, it usually has the same format of the remaining stanzas.

Nevertheless, I also performed some tests with other poems in the end, to test how the system could adapt, and the results were highly positive (as you can see in chapter 5).

1.4 The research

The main aspects I tried to achieve with my literature research has to do with the following: trying to understand the related work that had already been developed, seeing the text analysis rules that should be followed, knowing the main metric and prosody notions and, above all, finding the best tools to use in the process. On these tools, I will speak later in the chapter of Implementation (4).

1.5 Structure of the Dissertation

In this chapter, the theme and the motivation for the work are introduced. After this, in chapter 2, the literature is presented and the key terms and notions are referred. The system operations from the input to the output are explained in the chapter 3, in the next chapter (chapter 4 – Implementation) contains basic information about the tools that were used and the system's code itself. The chapter 5 has some examples of the

application of the system in different lyrics, while the conclusion (chapter 6) has the main ideas that this work lead to, as well as my personal evaluation of it. The future work is also described. All the written code is in the appendix (chapter 8).

2 State of the Art

2.1 Introduction

The initial idea for this project came from a major concept: to create a system that automatically generates musical contents based only on text analysis. Once I proposed myself to take the first step, developing a system to analyze a lyric and generate a metrical structure, I think that the best way to start this system could only be one: a good text analysis.

Unlike the conventional approach to text analysis, what's important to the system to extract from the text is not its meaning. At least not at this stage. What it really needs to extract is the phonetic and lexical information.

The analysis of the text is performed using a phonetic information dictionary, the CMUdict. This dictionary is an open-source machine-readable pronunciation dictionary for North American English that contains over 134,000 words and their pronunciations. It was originally created by the Speech Group at Carnegie Mellon University (CMU) for use in speech recognition research. Despite being used in most cases in speech recognition and generation systems, this is not the first approach to this tool as a way of analyzing musical lyrics, as I will explain later in the Related Work chapter (2.4).

2.2 Musical Rhythm and Meter

The terms meter and rhythm are fundamental in this project as the ultimate goal of the system is to create a musical rhythm based on the stress of the verses that the user enters. The basis for a definition of meter as a cognitive mechanism is the fact that some pulses in a musical rhythm are perceived as stronger than others (George Sioros, 2016). Rhythm and meter are different concepts, but that connect directly. And this connection is made by a cognitive process in the listener's mind. By identifying a repetitive pattern of pulses with different intensities in the rhythm, the listener mentally draws the limits of the metric structure, creating an expectation for the rest of the song. From here, composers can work with this expectation, in most cases keeping a constant metric as a way to please the listener. They can also, however, challenge this expectation by changing the meter,

which when well done can give even more pleasure to the listener. In this project, the metric structure was kept constant for the sake of simplicity of the system.

2.3 Prosody

Prosody is a dimension of linguistics that's concerned with the elements of language that are responsible for its acoustic and rhythmic parameters. Technically, it's a field that covers more the spoken component of the language than the written one. And that's exactly why it's useful to the goal of the project. As mentioned in the abstract, one of the purposes of a musical lyric is being recited, and that's the focus of this work. Hence my greatest interest in linguistics than in grammar.

Digging deeper, the characteristic that matters most to my system within the prosody is the lexical stress, which carries information about accentuation levels. The production and perception of this stress are affected by acoustic variables as duration, intensity and frequency.

As seen in the previous section, the metrical structure is the periodic alternation of strong and weak beats, usually forming a nested hierarchy of accent levels and as in speech, the accentuation of musical meter involves change in the acoustic variables. Because of this, musical events aligned with higher metrical levels usually receive strong accent, are emphasized by performers and are better remembered by listeners (Palmer & Kelly, 1992).

2.4 Related Work

The Computer Science Department of Brigham Young University has looked at the issue of the musical creation from lyrics and carried out some research on it. In "Automatic Generation of Melodic Accompaniments for Lyrics", Monteith, Martinez & Ventura (2012) describe a system for automatic composition of musical accompaniment for lyrics. Besides the rhythm, the system addresses the melody and the musical style too. Rhythms are generated based on patterns of syllabic stress in the lyrics. The system then attempts to find the best position for downbeats. The number of measures is always assumed as

four and only four syllables are selected to carry a downbeat. For each line, one hundred random assignments are created, being then scored to pick the best. This part of the project has several similarities to this one I'm presenting.

In "Lyric-Based Rhythm Suggestion", Eric Nichols (2009) describes a system whose algorithm is composed of three main components: a scoring function used to judge the relative success of candidate rhythms, a database of English pronunciation to determine syllable stress levels, and traditional search techniques to find highscoring rhythms in a large space of candidate rhythms. Beyond the lyrics, this system require two extra inputs: desired total duration and time signature.

3 From the Lyrics Input to the Metric Output

3.1 Introduction

This prototype I'm presenting can't read as we humans do, but it was developed to interpret numbers, so the original text must be converted to a phonetic and lexical representation of itself, based on numbers. This numbers are markers and identify different lexical stress levels on the words.

3.2 Lyrics Analysis

One of the most important flags of this project is to create a system that deals only with text, without further input of any kind. Thus, the end result rather than generated, it will be found. The introduced lyrics gives us several verses with many different forms, and one of them will stand out as the best, taking into account several factors.

The system's approach to the text is the most humane possible. From left to right, line by line, every word is individually understood and converted to a machine-readable format. Of course, only filled rows will be handled, while the remaining are markers between paragraphs. Every word of every line is crossed with a phonetic dictionary, which returns the information associated with the word. The text has now a different appearance, as shown in the Figure 1.

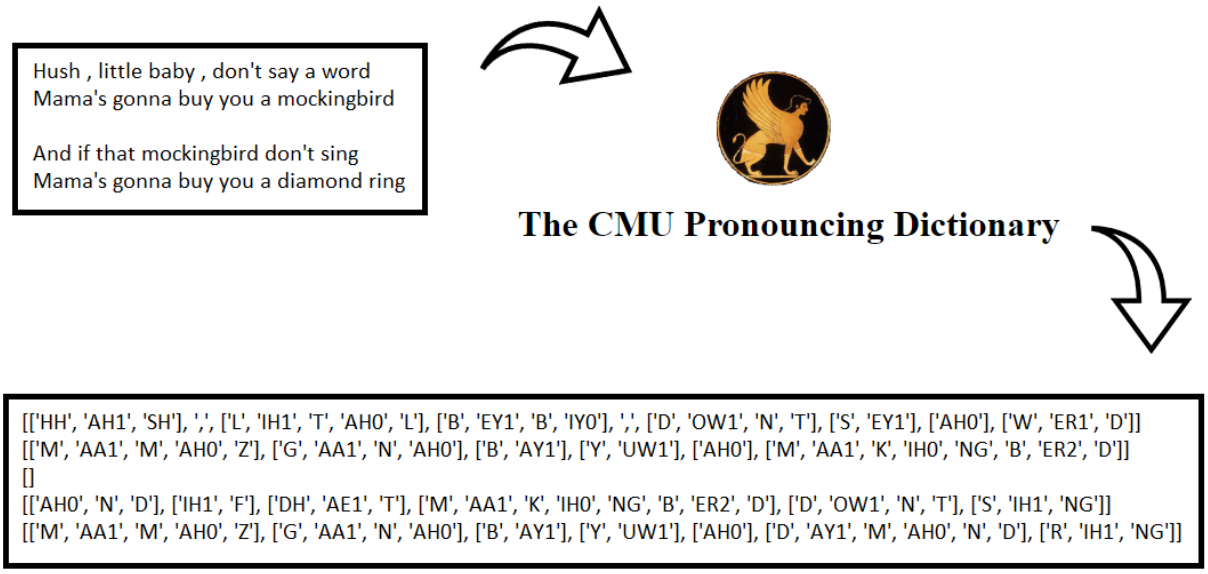


Figure 1: Conversion of the first two stanzas to its phonetic representation.

3.2.1 Lexical Stress extraction

The first resource to be used is the lexical stress. As it is possible to see in the Figure 1, some phones have a number at the end. Only the vowels contain this markers, which represent the lexical stress level. For the next step, this is the information that will be used.

The idea is to find a common metrical form. Or the most common possible, since it would be virtually impossible that all lines have the same accentuations in the same order. There are three markers of lexical stress in this dictionary:

0 - No Stress

1 - Primary Stress

2 - Secondary Stress

The 0 stands for vowels with a weak lexical stress and the 1 and 2 for vowels with a strong stress. If it's a compound word (as "mockingbird" in the Figure 1) the primary stress (1) is assigned to the leftmost word and the second word is assigned with a secondary stress (2), as it was established by the Compound Word Rule of English (Chomsky and Halle, 1968). There is no hypothesis that there is a 2 if not already there is a 1 in the same word. For each line of text, the markers will be counted and separated. The 1 and 2 are treated equally, because the system is only labeling the phones as weak (0) or strong (1 and 2).

3.3 Mapping of Prosodic Accents to the Metric Grid

For each line the stress markers are counted and a metric grid is generated. The number of beats is defined by the amount of strong stress markers in each line. Assuming at this first phase of the project the 4/4 as a standard meter, the duration of the meter will be the lowest power of 2 that is greater than the number of strong stress markers. Therefore, they will always be 4, 8, 16, 32, etc.

The metric grids are built by levels. The highest level, whose number of beats are defined by the duration of the meter, is hosting the 1's. Between each beat, spaces will be created to accommodate the 0's (this is the second level. It may be necessary to create more empty spaces, creating more levels).

For structural reasons, the last beat of each measure can't be filled, in order to always leave a minimum pause between each verse. To make sure that the first beat is always a strong stressed phone, for each line all the weak markers before the first strong will be removed and stored. These phones are the anacrusis (like an introduction of the verse), and will be always assigned to the end of the previous one. The number of strong accents will be distributed in a simple way. The first takes the first beat, the second is the second beat, and so on. In order to respect the rule of never occupy the last beat, if the number of strong accents is exactly equal to certain power of 2, the immediately higher will be used.

After properly distributed the strong accents, the weak stress markers are placed in the spaces between the beats. There are some rules that cannot be broken. The zeros

cannot occupy the first level beats, and they must be placed exactly between the strong accents they belong to. Although at this stage they are numbers, they represent vowels in words, so if the order is changed in the process, at the end the text won't make sense.

3.3.1 Generation of Metrical Templates for Prosody

The sequence of stress markers of each verse must be converted into a template to be tested. For this, the system applies the transformation explained in chapter 3.3. In Figure 2, we can see how a sequence of markers from a verse will be transformed into a metric template

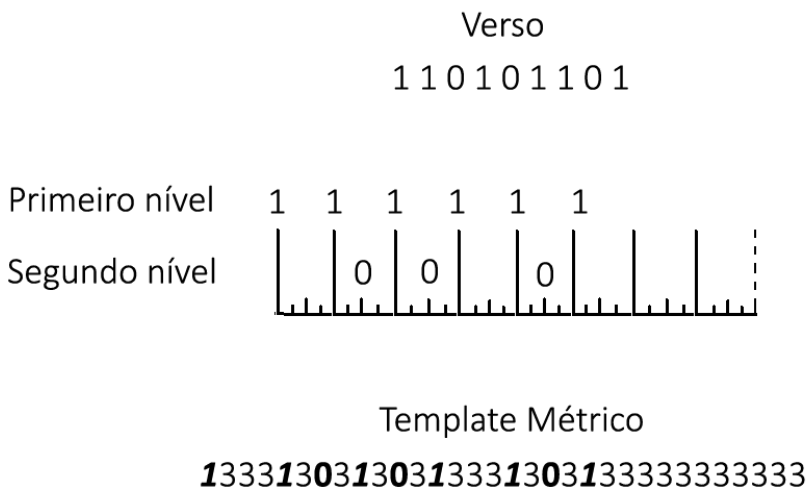


Figure 2: Distribution of the lexical stress markers of the first verse by the metric grid. Once that has six strong accents, the grid was built with eight beats on the first level (the first power of two bigger than six).

In order to be easier to use the templates for the purpose of comparison, a new value will be introduced (3) to represents the rests. From here all the templates have the same length. So, the first verse which until now was represented by the numbers of their accents (110101101), shall also contain 3's. These new values will take up all the blanks of the Figure 2. Thus, the first verse representation will be: "**1**333**1**3**0**3**1**3**0**3**1**333**1**3**0**3**1**33333333333".

3.3.2 Recursive Alignment of Verses in the Templates

After converting all the verses to metrical templates, it is time to start the alignments. I developed an algorithm that allows the system to do a recursive analysis to all templates, crossing each one with all the verses. And by crossing, I mean to distribute the lexical markers of each verse within the limits of each template. As each verse is different in general, several modifications must be made to each one so that it can adjust to the templates, giving us in most cases several results for each operation. There are also some rules on this operation. The first metric level of the template that's being tested is not modifiable. So, if it has six first level beats filled (as in Figure 2), all the strong stress markers of each verse must be aligned in this same six beats. If they are more, some of them are downgraded to the second level. If they are less, a rest is assigned to the template's filled beats that don't have any strong accent to its place. In both cases, all the possible alignment hypothesis are generated and stored in a multidimensional array.

Once the number of templates and the number of verses is set (which may not be the same, as I will explain later in this chapter), a multidimensional array is created to further accommodate the possibilities generated for each crossing between them. This array will be created with two dimensions, the first being the templates and the second the verses.

```
i,j=len(templatesMetrics),len(verseStress)
resultsArray=[[None for x in range(j)] for y in range(i)]
```

In this lines above, we can see the array being created and filled with empty dimensions (None means empty). y dimensions are created inside each x dimension, being x the number (range) of templates and y the number of verses.

Each time one runs the recursive operation, a third dimension is added to the y dimensions of the array, containing the alignment possibilities.

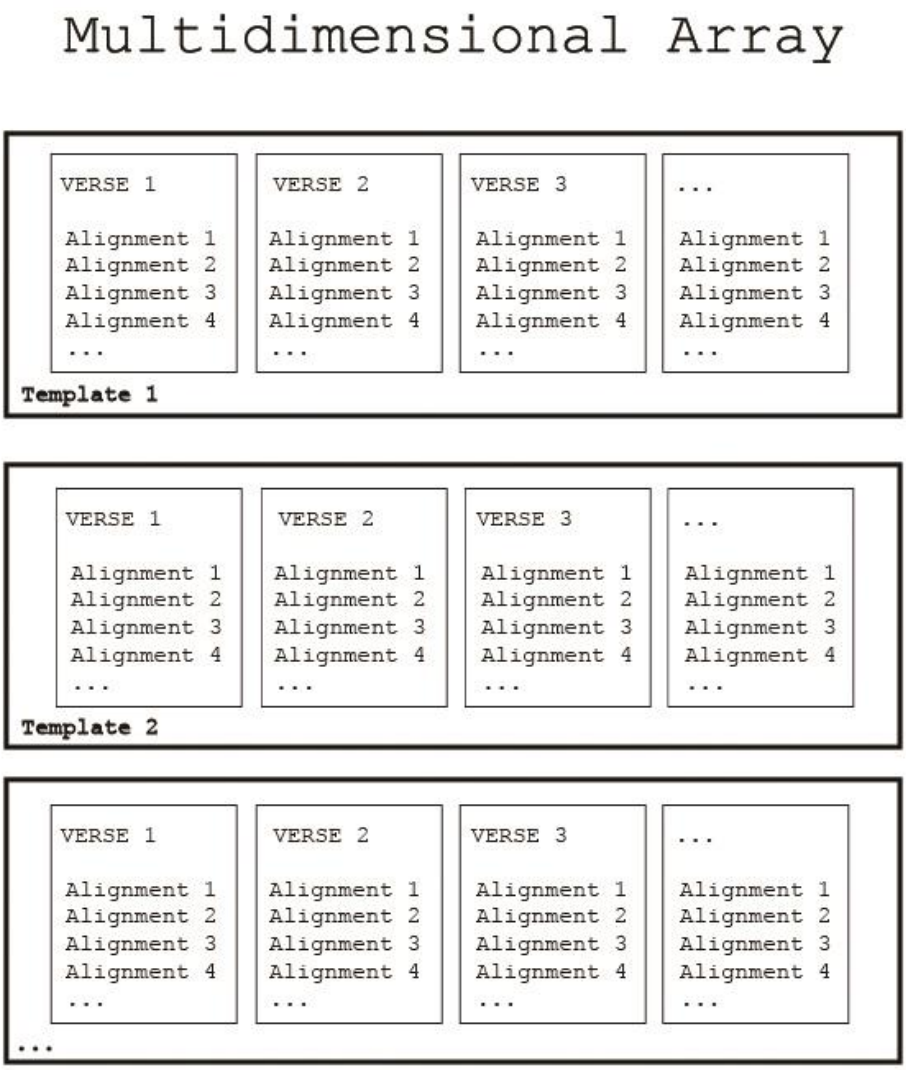


Figure 3: Exemplification of the multidimensional array construction. Each verse inside each template will store each alignment possibility.

After the full analysis is done, if one wants to see the alignment possibilities of the fifth verse in the second template, it can be accessed as follows: resultsArray[2][5].

The recursive function is performed n times, where n is the multiplication of the verses with the verses themselves. Although, technically, it may not be so because duplicate templates will be removed, increasing the efficiency of the system, since it does not need to repeat an entire process to get the same results (even if a duplicate template is removed, the verses will all remain, so the score is not affected). The function takes two inputs, the first being the template verse (X) and the second the verse to be tested within this template (Y).

```
for t in range(len(templatesMetrics)):
    for v in range(len(verseStress)):

rhythm[t][v]=recursive(templatesMetrics[t],verseStress[v])
```

In the first step of each operation, the strong accents are aligned. Let's imagine that the current template (X) has six strong accents. If the verse to be tested (Y) has eight, only six will stay in the upper level and 2 have to be set at the lower level. This will return several possibilities, because these two strong stresses will be placed in the second level of all possible ways.

To make it clear, I will present two examples of strong lexical markers distributions. The template is the same in both instances, although in the first case the verse has one strong marker less, while in the second has one more.

```
Template:  1 _ 0 _ 1 _ 0 _ 1 _ _ _ 1 _ _ _ 1 _ 0 _ 1 _ 0 _ 1 _ _ _ _ _ _ _
_
Verse:  1 1 0 1 0 1 1 0 1
Alignment possibilities:
1 _ _ _ 3 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1_ _ _ _ _ _ _
1 _ _ _ 1 _ _ _ 3 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1_ _ _ _ _ _ _
1 _ _ _ 1 _ _ _ 1 _ _ _ 3 _ _ _ 1 _ _ _ 1 _ _ _ 1_ _ _ _ _ _ _
1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 3 _ _ _ 1 _ _ _ 1_ _ _ _ _ _ _
1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 3 _ _ _ 1_ _ _ _ _ _ _
```

```
Template:  1 _ 0 _ 1 _ 0 _ 1 _ _ _ 1 _ _ _ 1 _ 0 _ 1 _ 0 _ 1 _ _ _ _ _ _ _
_
Verse:  1 1 1 1 0 1 0 1 0 1 0 0 1
Alignment possibilities:
1 _ 1 _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1_ _ _ _ _ _ _
1 _ _ _ 1 _ 1 _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1_ _ _ _ _ _ _
1 _ _ _ 1 _ _ _ 1 _ 1 _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1_ _ _ _ _ _ _
1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ 1 _ 1 _ _ _ 1 _ _ _ 1_ _ _ _ _ _ _
1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ 1 _ 1 _ _ _ 1_ _ _ _ _ _ _
1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ _ _ 1 _ 1 _ 1_ _ _ _ _ _ _
```

In the first operation of the recursive function, the input X is the template[0] and the input Y is the verse[0], and only one alignment is returned because they are the same. The second operation is done with the template[0] being X again, but now the input Y is the verse[1], and it can return several possible alignments, depending on the difference of lexical stress markers between them. Once all verses are aligned with the template[0], the operation is repeated to all the other templates.

The greater the difference of strong accents from Y to X, more alignments will be returned. Once completed this phase, the systems has a set of alignments (or just one, in some cases) for each crossing, but only with the strong stress markers. It is now time to place the weak stress markers in all this alignments.

This is done by a different function that receives also two inputs, but not the template X anymore. The first input is each alignment of the previous function and the second is the Y they belong to. As mentioned earlier in the chapter explaining the construction of metric templates, zeros cannot occupy the spaces on the upper level, even if the verse Y has fewer strong markers than the template X. Each weak stress markers (0) of Y is analyzed individually and the strong markers (1) that precede it are counted. Then, for each alignment, the same number of 1's is counted and the 0 is placed in the next second level space. If there are consecutive weak stress markers in the verse Y, they will have the same number of preceding strong markers. In this case, a new lower level is created (if it does not already exist) to accommodate these zeros.

3.3.3 Selection of the Best Template

It is now time to choose the best template, starting by first selecting the best alignment possibility for each second dimension of the array. Since we now have an array organized as follows: **[template] [verse] [alignment possibilities]**, we need to convert it to this: **[template] [verse] [best alignment for this crossing]**. For each of the possibilities presented, a score will be assigned. The more changes you need to make for a verse fit a template, more points are added to its score. Furthermore, different types of changes will add a different numbers of points.

The function “Scoring” receives a first dimension template and each of the possibilities from the third dimension regarding this same template and each of the verses. This can be a bit confusing, but I will try to explain it in Figure 4.

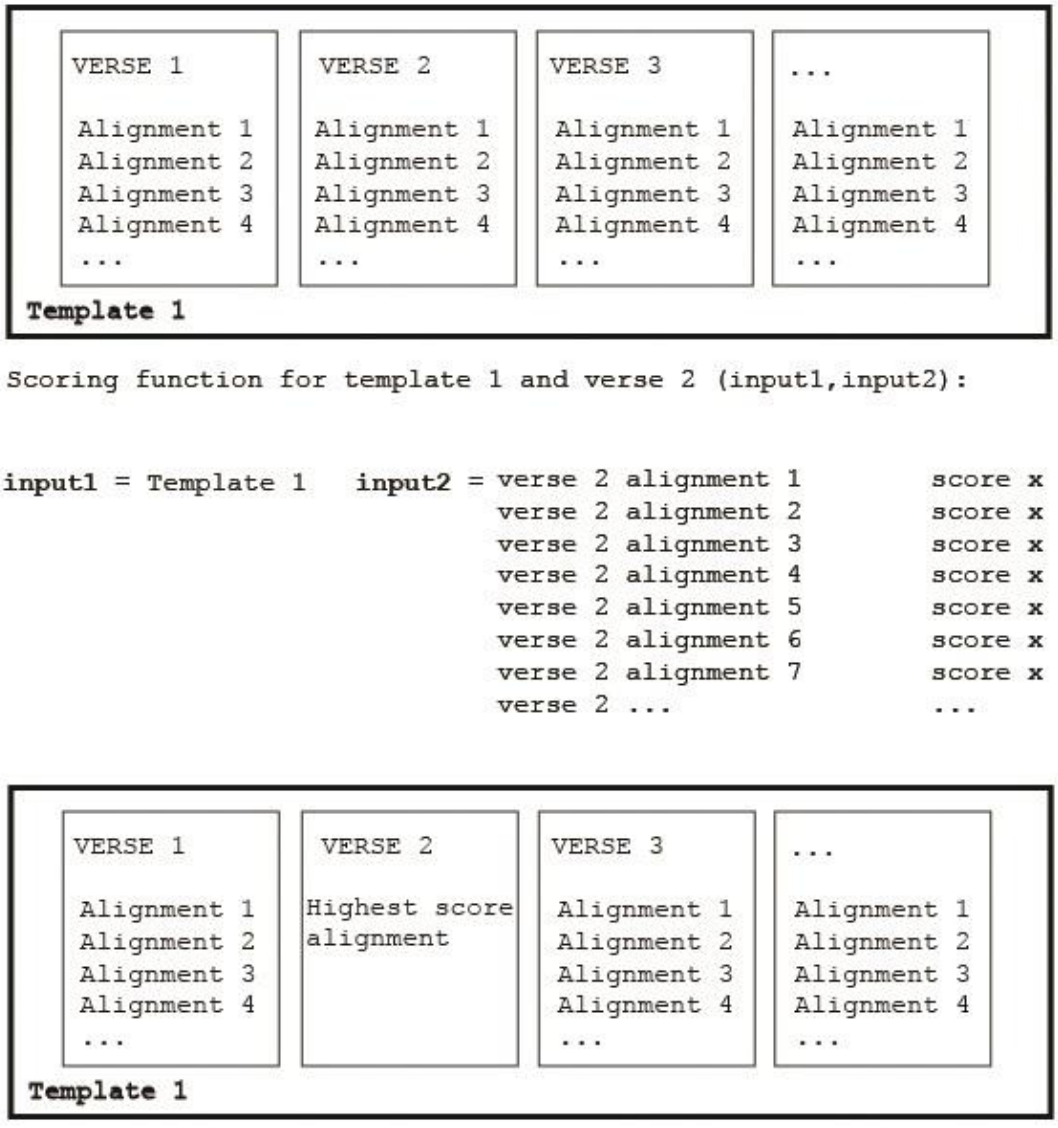


Figure 4: Explanation of the scoring function process with verse 2 of template 1.

To perform the scoring function in a recursive way, it is triggered the same way "recursive function" was, and the results are stored in an identical array (this one is called "score", but has the same structure):

```
for t in range(len(templatesMetrics)):
    for p in range(len(verseStress)):

score[t][p]=scoring(templatesMetrics[t],resultsArray[t][p])
```

The first step in this function is to convert both the current template and alignment possibility to the same length. This is done by adding rests to the shortest (or adding a new layer, it's basically the same). Different metric layers have different weights (**W**) - the higher the level is, the heavier its weight will be - on this scoring process. After converted to the same size, they can be compared each unit at a time. If they don't match, it means that was necessary to change the original template in order to align the verse, and **N** points are assigned to the score of that possibility. Depending on the level to which they belong, the assigned points will be defined by **(N*W)**. After some tests, **N** was set to be equivalent to 25. If the element to which the value is assigned belongs to level 1, the number of points will be multiplied by 4 (**W** = 4). If its place is at level 2, **W** = 2. If none of the cases, then the weight will be 0 and it will be assigned 25 points.

For each alignment, the possibility with lowest score will be selected and stand out as the best. So, we now have:

```
resultsArray[template][verse][minScore(alignment)]
```

The same process will be used to find the best template: the score of each template will be defined by the sum of the selected possibility for each of its alignments with all the verses, and the template with the lowest score will be selected.

3.4 Results Analysis and Output

After selecting the best template, the metric structure is created, which will consist on the element with the lowest score of the third dimension related to that same template, as follows:

```
resultsArray[best(template)][each(verse)][minScore(alignment)]
```

Finally, the numbers of each row must be substituted by its respective phonemes again. For converting verses into a metrical representation some phones (especially consonants) were omitted by not contain any lexical stress marker. These phones will now be grouped with the more indicated lexical stressed phoneme for itself. Here is the two first verses with all the phonemes.

```
[['HH', 'AH1', 'SH'], ['L', 'IH1', 'T', 'AH0', 'L'], ['B', 'EY1', 'B',
'IY0'], ['D', 'OW1', 'N', 'T'], ['S', 'EY1'], ['AH0'], ['W', 'ER1', 'D']]

[['M', 'AA1', 'M', 'AH0', 'Z'], ['G', 'AA1', 'N', 'AH0'], ['B', 'AY1'],
['Y', 'UW1'], ['AH0'], ['M', 'AA1', 'K', 'IH0', 'NG', 'B', 'ER2', 'D']]
```

In the first word, all the non-stressed phones will be assigned to the stressed one. But in the second, the process is a little bit more complicated.

This process is executed based on a simple algorithm developed for this purpose, which is explained in the Figure 5. Each word is treated individually and for each of its elements (phone) is performed an operation. A storage is created to hold some be holding some elements during the process. The first step is to check if the phone is stressed or not. If it is, it's assigned to its respective stress marker place. Every time a stressed phone is assigned and there is content in the storage, it is added concatenated to the phone that's going to be assigned. If the phone is not stressed, two things can happen: if the storage is empty, this phone is sent to it. If there's already some phone in the storage, the existing phone is concatenated to the last assigned element and the new phone is taking its place

in the storage. Only if there are no assigned phonemes yet, the storage can take both phonemes in it, and will treat them as one.

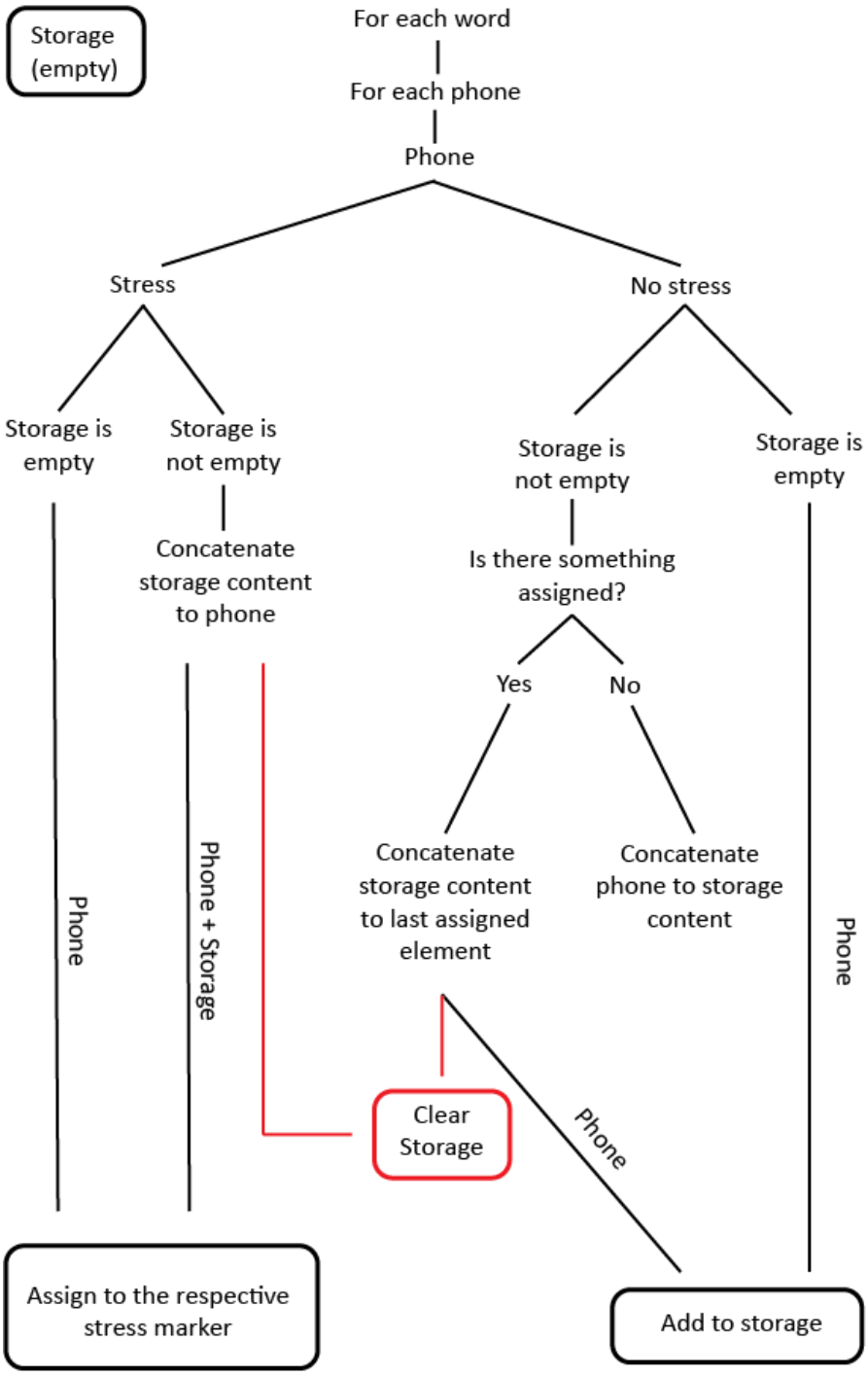


Figure 5: Phones assignment algorithm explanation.

4 Implementation

4.1 CMUdict

CMUdict is a pronunciation dictionary for North American English. Its phoneme (or more accurately, phone) set is based on the ARPAbet symbol set developed for speech recognition uses and contains 39 phonemes, as we can see below.

Phoneme	Example	Translation
AA	odd	AA D
AE	at	AE T
AH	hut	HH AH T
AO	ought	AO T
AW	cow	K AW
AY	hide	HH AY D
B	be	B IY
CH	cheese	CH IY Z
D	dee	D IY
DH	thee	DH IY
EH	Ed	EH D
ER	hurt	HH ER T
EY	ate	EY T
F	fee	F IY
G	green	G R IY N
HH	he	HH IY
IH	it	IH T
IY	eat	IY T
JH	gee	JH IY
K	key	K IY
L	lee	L IY
M	me	M IY
N	knee	N IY
NG	ping	P IH NG
OW	oat	OW T
OY	toy	T OY
P	pee	P IY
R	read	R IY D
S	sea	S IY
SH	she	SH IY
T	tea	T IY
TH	theta	TH EY T AH
UH	hood	HH UH D
UW	two	T UW
V	vee	V IY
W	we	W IY
Y	yield	Y IY L D
Z	zee	Z IY
ZH	seizure	S IY ZH ER

For each word, this lexicon provides a list of phonetic codes—distinct labels for each contrastive sound—known as phones. The following example shows some entries of the dictionary.

```
>>> entries = nltk.corpus.cmudict.entries()
>>> len(entries)
127012
>>> for entry in entries [39943:39947]:
... print entry
...
('fir', ['F', 'ER1'])
('fire', ['F', 'AY1', 'ER0'])
('firearm', ['F', 'AY1', 'ER0', 'AA2', 'R', 'M'])
('firearms', ['F', 'AY1', 'ER0', 'AA2', 'R', 'M', 'Z'])
('fireball', ['F', 'AY1', 'ER0', 'B', 'AO2', 'L'])
```

4.2 Natural Language Toolkit (NLTK)

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources, along with a suite of text processing libraries. It defines an infrastructure that can be used to build Natural Language Processing programs in Python. It provides basic classes for representing data relevant to natural language processing; standard interfaces for performing tasks such as part-of-speech tagging, syntactic parsing, and text classification; and standard implementations for each task that can be combined to solve complex problems.

NLTK comes with extensive documentation. In addition to its book, the website at http://www.nltk.org/ provides API documentation that covers every module, class, and function in the toolkit, specifying parameters and giving examples of usage. The website also provides many HOWTOs with extensive examples and test cases, intended for users, developers, and instructors.

This toolkit includes the CMU Pronouncing Dictionary for U.S. English in its corpora, which made the process of this project a lot easier and fast.

4.3 Python

All Python releases are open-source and it has a simple and pragmatic syntax, easy to learn and to use. Other important features are its dynamic type system and the large and comprehensive standard library.

4.3.1 Brief Explanation of Python Code

In this section, the code will be briefly explained.

- **First text analysis and extraction of the stress markers (lines 1-74) –** All the words are crossed with the dictionary and their phonetic information is added to [resultLine]. In the end of parsing each line, it appends the new line with all the phonemes to [firstAnalysis]. If some of them is not found, it returns the word along with the message: “word not recognized”.

- **Cleaning up the verses: finding the anacrusis and removing their stress markers (lines 75-108) –** For each verse, the system checks if there is an anacrusis and how many stress markers belong to it (In other words, how many weak stress markers appear before the first strong one). All the verses (without anacrusis) are added to [verseStress].

- **Creating the templates (lines 109-140) –** The function finalTemplate(x) receives each of the sequences of stress markers of verses and from them creates the metric templates. It finds the power of two of the amount of strong markers and creates a metric grid with that duration. Between each beat it creates a new space, rising a lower metric level. Strong markers are distributed by the beats and weak markers are placed on the lower level. All the templates are appended to [templatesMetrics].

- **Creating all the alignments possibilities for the templates (lines 141-290) –** recursive function takes all the templates and for each of them intersects with all the verses, generating several possibilities for each alignment.

Within this function there are other functions that distribute the different stress markers. All the results are saved in a multidimensional array.

- **Selecting the best template (lines 291-345) –** the scoring function takes each possibility hosted in the array and assigns it a score, based on how much its alignment respects the structure of the template to which it belongs. Based on this score it is chosen for each template the best alignment of each verse, and their scores are combined all together to give the template a general score. The template with best overall score is selected as the best.

- **Triggering all the functions, creation of the multidimensional arrays and distribution of the lyrics by the metric grid and output (lines 346-500) –** Once the functions are created, it's necessary to trigger them. The systems creates the multidimensional arrays and then fill them by sending all the templates and verses to the functions. This is what triggers the system. After the best template is chosen, the system returns the result to each verse. The non-stressed phonemes are grouped with the stressed ones they belong to (by the assign function), and each group takes its place in metric structure, making a visual presentation to the user.

5 Application to six lyrics/poems

In order to analyze the system performance, some tests were conducted with different lullabies and children's poems. Due to the above referenced limitations, mainly in terms of vocabulary (many lullabies contain onomatopoeia) and lack of metric coherence of the texts, some were rejected. Those that were correctly analyzed yielded very interesting results, as I will show through a more musically conventional representation of these results. The raw results are in appendix chapter (8).

5.1 Lullabies

5.1.1 All the Pretty Horses

- **Lyrics**

```
Hush a bye don't you cry
Go to sleep my little baby
When you wake you shall have
All the pretty little horses
Black and bays checkered grays
All the pretty little horses
Hush a bye don't you cry
Go to sleep my little baby
Hush a bye don't you cry
Go to sleep my little baby
When you wake you shall have
All the pretty little horses
```

- **Stress markers metric distribution output**

```
1_0_1___1___1___1_______________
1_1_1___1___1_0_1_0_____________
1_1_1___1___1___1_______________
1_0_1___0___1_0_1_0_____________
1_0_1___3___1_0_1_______________
1_0_1___0___1_0_1_0_____________
1_0_1___1___1___1_______________
1_1_1___1___1_0_1_0_____________
1_0_1___1___1___1_______________
1_1_1___1___1_0_1_0_____________
1_1_1___1___1___1_______________
1_0_1___0___1_0_1_0_____________
```

- **Phones Assignment to the markers**

```
HH|AH|SH._AH._B|AY.___D|OW|N|T.___Y|UW.___K|R|AY._______________
G|OW._T|UW._S|L|IY|P.___M|AY.___L|IH._T|AH|L._B|EY.__B|IY._____________
W|EH|N._Y|UW._W|EY|K.___Y|UW.___SH|AE|L.___HH|AE|V._______________
AO|L._DH|AH._P|R|IH.___T|IY.___L|IH._T|AH|L._HH|AO|R._S|AH|Z._____________
B|L|AE|K._AH|N|D._B|EY|Z._______CH|EH._K|ER|D._G|R|EY|Z._______________
AO|L._DH|AH._P|R|IH.___T|IY.___L|IH._T|AH|L._HH|AO|R._S|AH|Z._____________
HH|AH|SH._AH._B|AY.___D|OW|N|T.___Y|UW.___K|R|AY._______________
G|OW._T|UW._S|L|IY|P.___M|AY.___L|IH._T|AH|L._B|EY.__B|IY._____________
HH|AH|SH._AH._B|AY.___D|OW|N|T.___Y|UW.___K|R|AY._______________
G|OW._T|UW._S|L|IY|P.___M|AY.___L|IH._T|AH|L._B|EY.__B|IY._____________
W|EH|N._Y|UW._W|EY|K.___Y|UW.___SH|AE|L.___HH|AE|V._______________
AO|L._DH|AH._P|R|IH.___T|IY.___L|IH._T|AH|L._HH|AO|R._S|AH|Z._____________
```

- **Musical Notation Transcription**

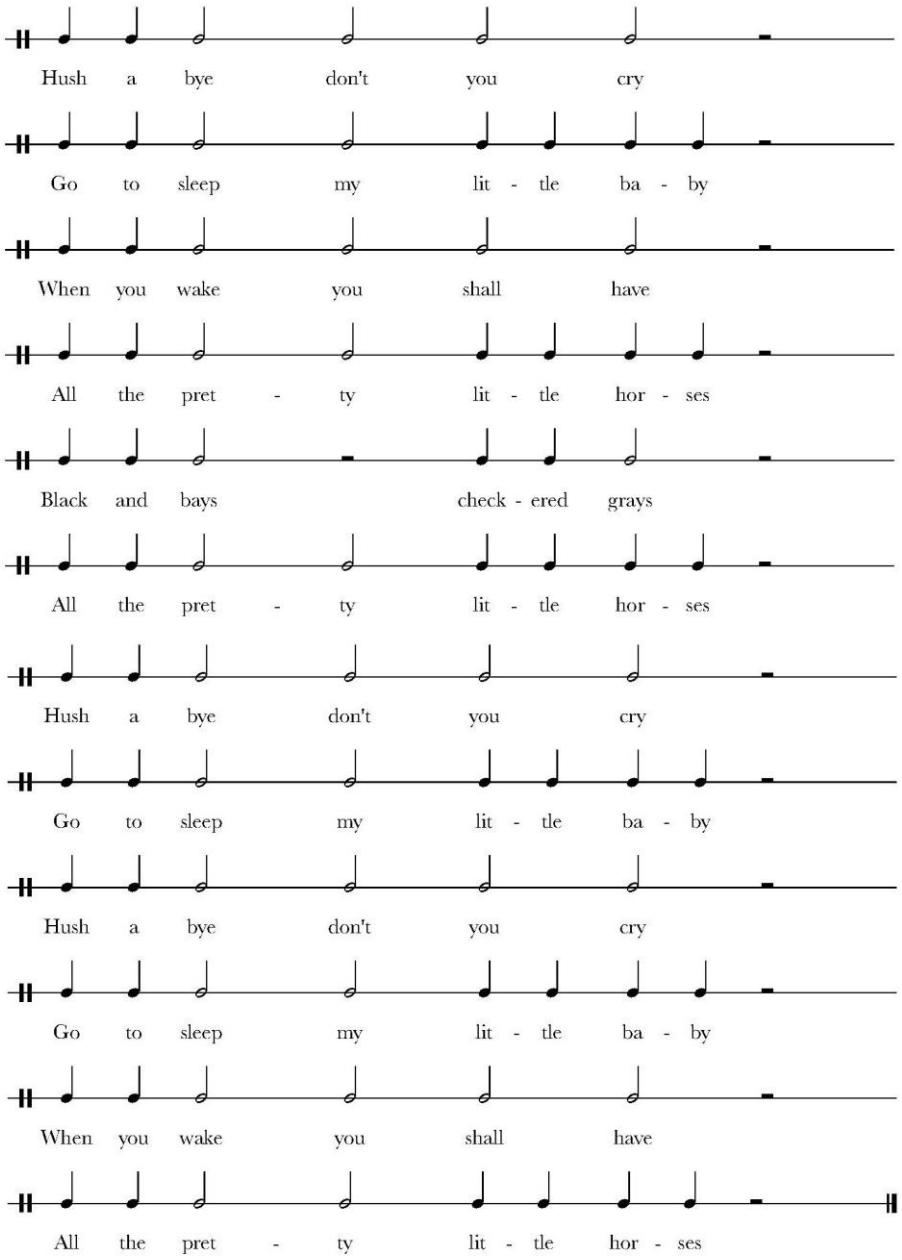


Figure 6: Musical Notation Transcription of the results for the lyric "All Pretty Horses".

5.1.2 Hush, Little Baby

- **Lyrics**

```
Hush little baby don't say a word
Mama's gonna buy you a mockingbird

And if that mockingbird don't sing
Mama's gonna buy you a diamond ring

And if that diamond ring turns brass
Mama's gonna buy you a looking glass

And if that looking glass gets broke
Mama's gonna buy you a billy goat

And if that billy goat don't pull
Mama's gonna buy you a cart and bull

And if that cart and bull turn over
Mama's gonna buy you a dog named Rover

And if that dog named Rover won't bark
Mama's gonna buy you a horse and cart

And if that horse and cart fall down
Well you'll still be the sweetest little baby in town
```

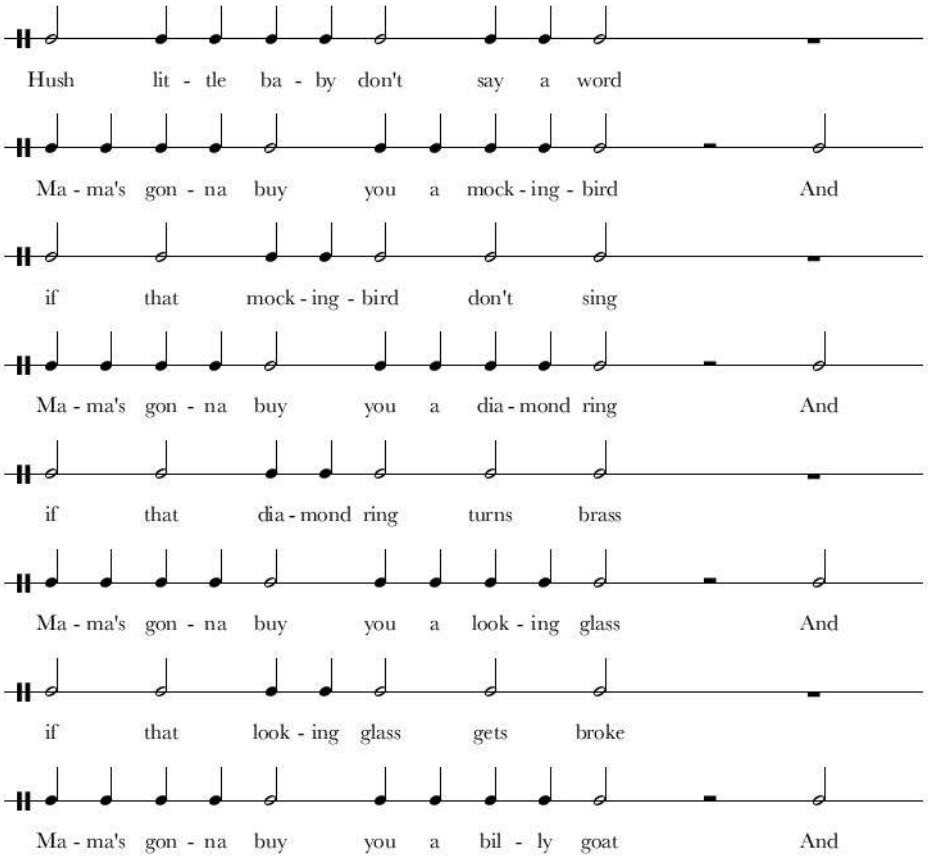
- **Stress markers metric distribution output**

```
1___1_0_1_0_1___1_0_1___________
1_0_1_0_1___1_0_1_0_1_______0___
1___1___1_0_1___1___1___________
1_0_1_0_1___1_0_1_0_1_______0___
1___1___1_0_1___1___1___________
1_0_1_0_1___1_0_1_0_1_______0___
1___1___1_0_1___1___1___________
1_0_1_0_1___1_0_1_0_1_______0___
1___1___1_0_1___1___1___________
1_0_1_0_1___1_0_1_0_1_______0___
1___1___1_0_1___1___1__0________
1_0_101_1_0_1___1___1__0____0___
1___1___1___1___101_1___________
1_0_1_0_1_1_1_0_1_0_1_______0___
1___1___1_0_1___1___1___________
1___1_1_1_0_10101_0_1___________
```

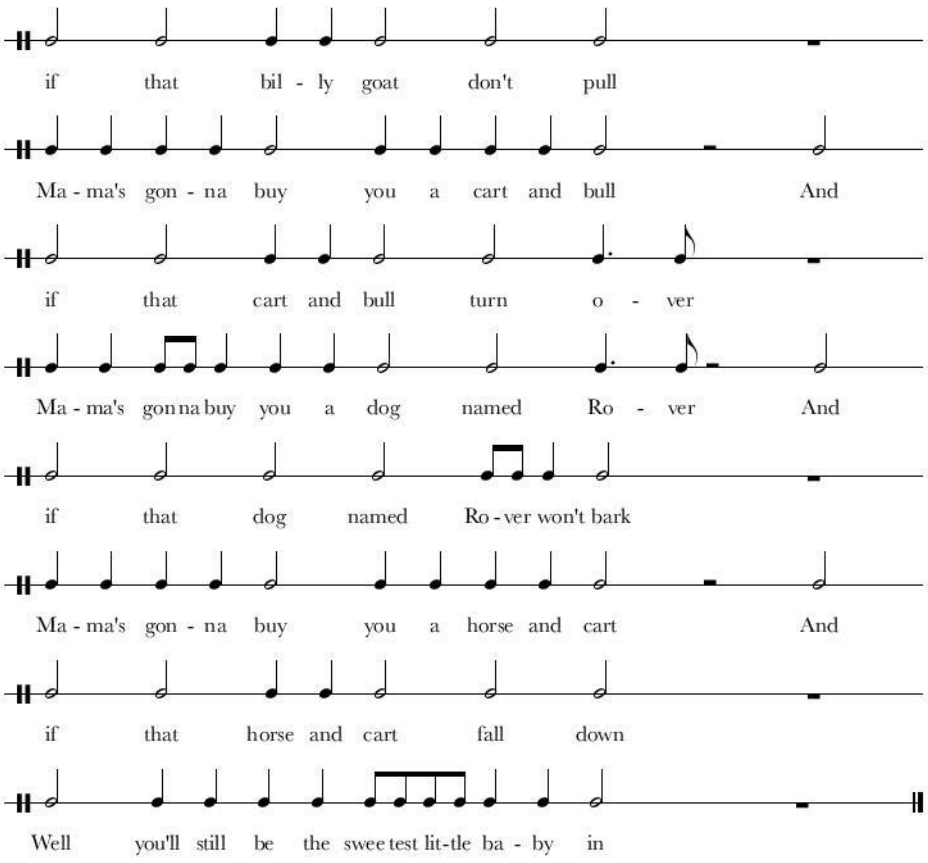
- **Phones Assignment to the markers**

```
HH|AH|SH._____L|IH._T|AH|L._B|EY._B|IY._D|OW|N|T._____S|EY._AH._W|ER|D._____________

M|AA._M|AH|Z._G|AA._N|AH._B|AY._____Y|UW._AH._M|AA._K|IH|NG._B|ER|D._______AH0|N|D._____

IH|F._____DH|AE|T._____M|AA._K|IH|NG._B|ER|D._____D|OW|N|T._____S|IH|NG._____________

M|AA._M|AH|Z._G|AA._N|AH._B|AY._____Y|UW._AH._D|AY._M|AH|N|D._R|IH|NG._______AH0|N|D._____

IH|F._____DH|AE|T._____D|AY._M|AH|N|D._R|IH|NG._____T|ER|N|Z._____B|R|AE|S._____________

M|AA._M|AH|Z._G|AA._N|AH._B|AY._____Y|UW._AH._L|UH._K|IH|NG._G|L|AE|S._______AH0|N|D._____

IH|F._____DH|AE|T._____L|UH._K|IH|NG._G|L|AE|S._____G|EH|T|S._____B|R|OW|K._____________

M|AA._M|AH|Z._G|AA._N|AH._B|AY._____Y|UW._AH._B|IH._L|IY._G|OW|T._______AH0|N|D._____

IH|F._____DH|AE|T._____B|IH._L|IY._G|OW|T._____D|OW|N|T._____P|UH|L._____________

M|AA._M|AH|Z._G|AA._N|AH._B|AY._____Y|UW._AH._K|AA|R|T._AH|N|D._B|UH|L._______AH0|N|D._____

IH|F._____DH|AE|T._____K|AA|R|T._AH|N|D._B|UH|L._____T|ER|N._____OW.__V|ER._________

M|AA._M|AH|Z._G|AA.N|AH.B|AY._Y|UW._AH._D|AO|G._____N|EY|M|D._____R|OW.__V|ER._____AH0|N|D.__
-

IH|F._____DH|AE|T._____D|AO|G._____N|EY|M|D._____R|OW.V|ER.W|OW|N|T._B|AA|R|K._____________

M|AA._M|AH|Z._G|AA._N|AH._B|AY._____Y|UW._AH._HH|AO|R|S._AH|N|D._K|AA|R|T._______AH0|N|D._
--

IH|F._____DH|AE|T._____HH|AO|R|S._AH|N|D._K|AA|R|T._____F|AO|L._____D|AW|N._____________

W|EH|L._____Y|UW|L._S|T|IH|L._B|IY._DH|AH._S|W|IY.T|AH|S|T.L|IH.T|AH|L.B|EY._B|IY._IH|N._T
|AW|N._________
```

- **Musical Notation Transcription**



2


if that bil - ly goat don't pull
Ma - ma's gon - na buy you a cart and bull And
if that cart and bull turn o - ver
Ma - ma's gonna buy you a dog named Ro - ver And
if that dog named Ro - ver won't bark
Ma - ma's gon - na buy you a horse and cart And
if that horse and cart fall down
Well you'll still be the swee test lit-tle ba - by in

5.2 Children's Poetry

5.2.1 All Things Bright and Beautiful

- **Lyrics**

```
All things bright and beautiful
All creatures great and small
All things wise and wonderful
The Lord God made them all

Each little flower that opens
Each little bird that sings
He made their glowing colors
He made their tiny wings

The purple headed mountain
The river running by
The sunset and the morning
That brightens up the sky

The cold wind in the winter
The pleasant summer sun
The ripe fruits in the garden
He made them every one

He gave us eyes to see them
And lips that we might tell
How great is God Almighty
Who has made all things well
```

- **Stress markers metric distribution output**

```
1___3___1___1_0_1_0_____________
1___1___0___1_0_1_______________
1___3___1___1_0_1_0___________0_
1___1___1___1___1_______________
1___1_0_1_0_1___1_0_____________
1___1_0_1___1___1_______________
1___1___1___1_0_1_0_____________
1___1___1___1_0_1_____________0_
1___3_0_3___1_0_1_0___________0_
1___3_0_3___1_0_1_____________0_
1___3___3___1_0_1_0_____________
1___1___0___1_0_1_____________0_
1___3___3___1_0_1_0___________0_
1___3_0_3___1_0_1_____________0_
1___3___3___1_0_1_0_____________
1___1___1___1_0_1_______________
1___1_1_1___1_1_1_____________0_
1___1___1___1___1_______________
1___1___1___1_0_1_0_____________
1___1___1___1_1_1_______________
```

- **Phones Assignment to the markers**

AO|L._____TH|IH|NG|Z.___B|R|AY|T._AH|N|D._B|Y|UW._T|AH._____

AO|L.___K|R|IY.___CH|ER|Z.___G|R|EY|T._AH|N|D._S|M|AO|L._____

AO|L._____TH|IH|NG|Z.___W|AY|Z._AH|N|D._W|AH|N._D|ER._____DH|AH0.__

L|AO|R|D.___G|AA|D.___M|EY|D.___DH|EH|M.___AO|L._____

IY|CH.___L|IH._T|AH|L._F|L|AW._ER._DH|AE|T.___OW.__P|AH|N|Z._____

IY|CH.___L|IH._T|AH|L._B|ER|D.___DH|AE|T.___S|IH|NG|Z._____

HH|IY.___M|EY|D.___DH|EH|R.___G|L|OW._IH|NG._K|AH.__L|ER|Z._____

HH|IY.___M|EY|D.___DH|EH|R.___T|AY._N|IY._W|IH|NG|Z._____DH|AH0.__

P|ER._____P|AH|L._____HH|EH._D|AH|D._M|AW|N._T|AH|N._____DH|AH0.__

R|IH._____V|ER._____R|AH._N|IH|NG._B|AY._____DH|AH0.__

S|AH|N._____S|EH|T._AH|N|D._DH|AH._M|AO|R._____

DH|AE|T.___B|R|AY.___T|AH|N|Z.___AH|P._DH|AH._S|K|AY._____DH|AH0.__

K|OW|L|D._____W|AY|N|D._IH|N._DH|AH._W|IH|N._____DH|AH0.__

P|L|EH._____Z|AH|N|T._____S|AH._M|ER._S|AH|N._____DH|AH0.__

R|AY|P._____F|R|UW|T|S._IH|N._DH|AH._G|AA|R._____

HH|IY.___M|EY|D.___DH|EH|M.___EH._V|ER._IY._____

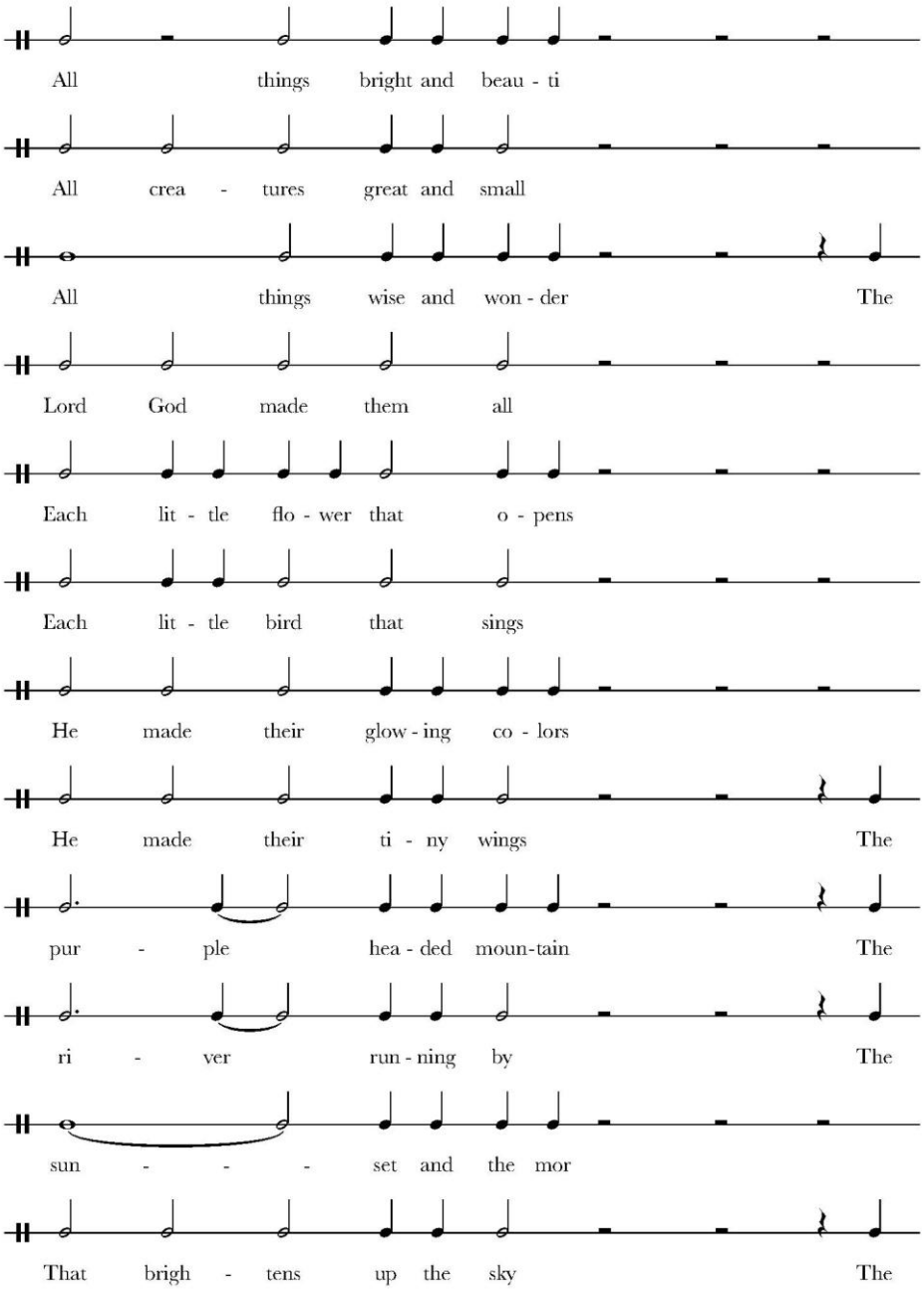
HH|IY.___G|EY|V._AH|S._AY|Z.___T|UW._S|IY._DH|EH|M._____AH0|N|D.__

L|IH|P|S.___DH|AE|T.___W|IY.___M|AY|T.___T|EH|L._____

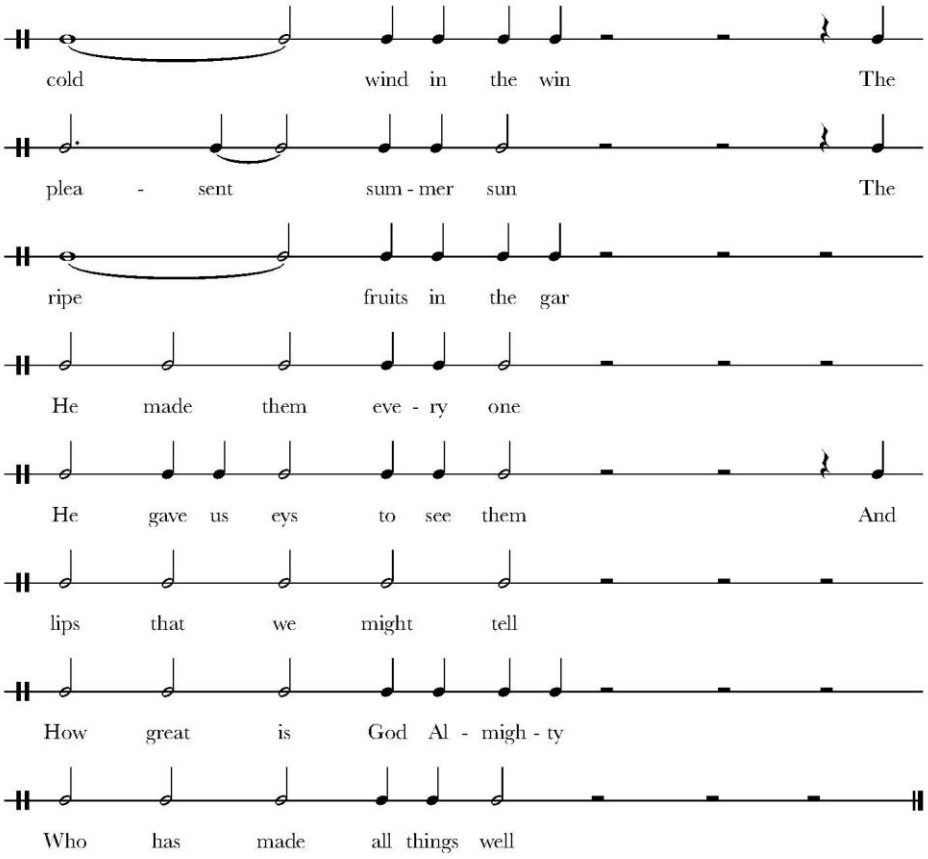
HH|AW.___G|R|EY|T.___IH|Z.___G|AA|D._AO|L._M|AY.__T|IY._____

HH|UW.___HH|AE|Z.___M|EY|D.___AO|L._TH|IH|NG|Z._W|EH|L._____

- **Musical Notation Transcription**



2


cold wind in the win The
plea - sent sum - mer sun The
ripe fruits in the gar
He made them eve - ry one
He gave us eys to see them And
lips that we might tell
How great is God Al - migh - ty
Who has made all things well

5.2.2 Bed in Summer

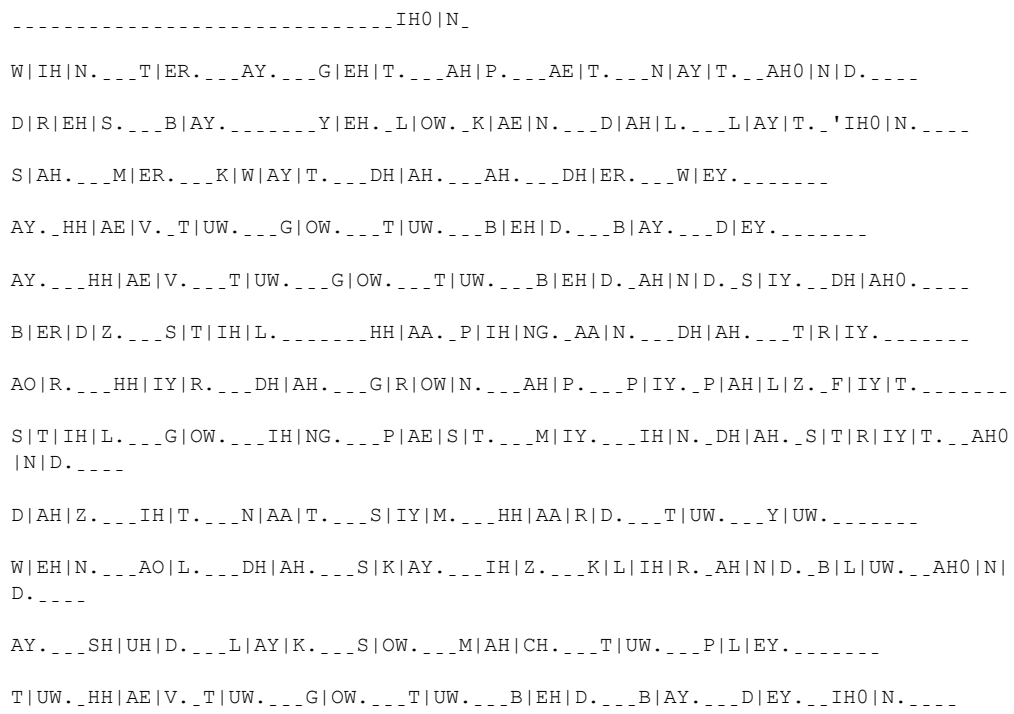
- **Lyrics**

```
In Winter I get up at night
And dress by yellow candle light
In Summer quite the other way
I have to go to bed by day

I have to go to bed and see
The birds still hopping on the tree
Or hear the grown up people's feet
Still going past me in the street

And does it not seem hard to you
When all the sky is clear and blue
And I should like so much to play
To have to go to bed by day
```

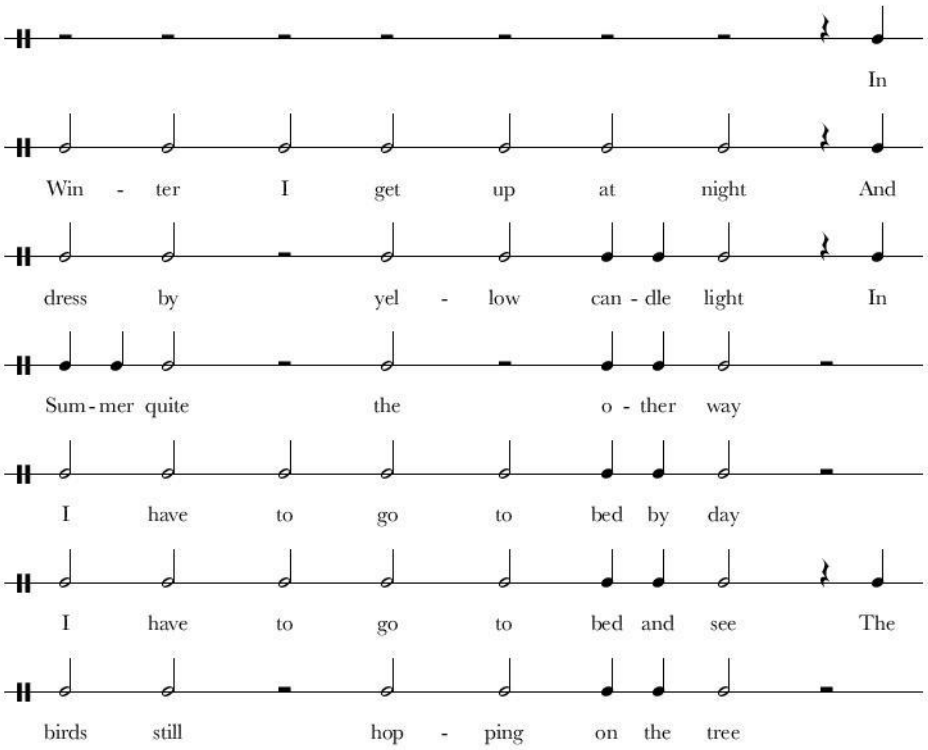
- **Stress markers metric distribution output**

```
------------------------------0_
1___0___1___1___1___1___1_____0_
1___1___3___1___0___1_0_1_____0_
1_0_1___3___0___3___1_0_1_______
1___1___1___1___1___1_1_1_______
1___1___1___1___1___1_0_1_____0_
1___1___3___1___0___1_0_1_______
1___1___0___1___1___1_0_1_______
1___1___0___1___1___0_0_1_____0_
1___1___1___1___1___1___1_______
1___1___0___1___1___1_0_1_____0_
1___1___1___1___1___1___1_______
1___1___1___1___1___1_1_1_______
```

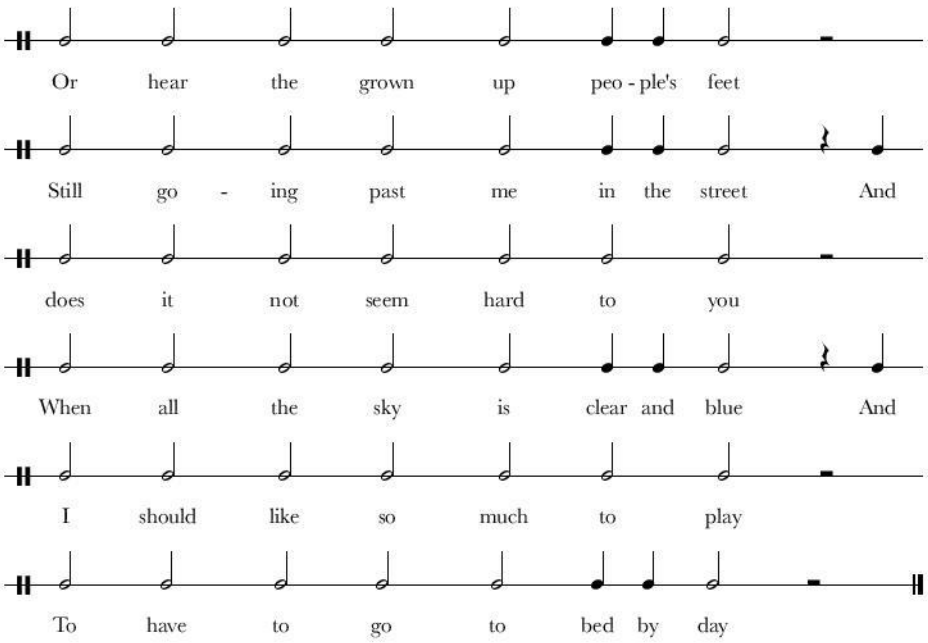
- **Phones Assignment to the markers**



- **Musical Notation Transcription**



2


Or hear the grown up peo - ple's feet
Still go - ing past me in the street And
does it not seem hard to you
When all the sky is clear and blue And
I should like so much to play
To have to go to bed by day

5.2.3 The Fieldmouse

- **Lyrics**

Where the acorn tumbles down
Where the ash tree sheds its berry
With your fur so soft and brown
With your eye so round and merry

Scarcely moving the long grass
Fieldmouse I can see you pass
Little thing in what dark den
Lie you all the winter sleeping

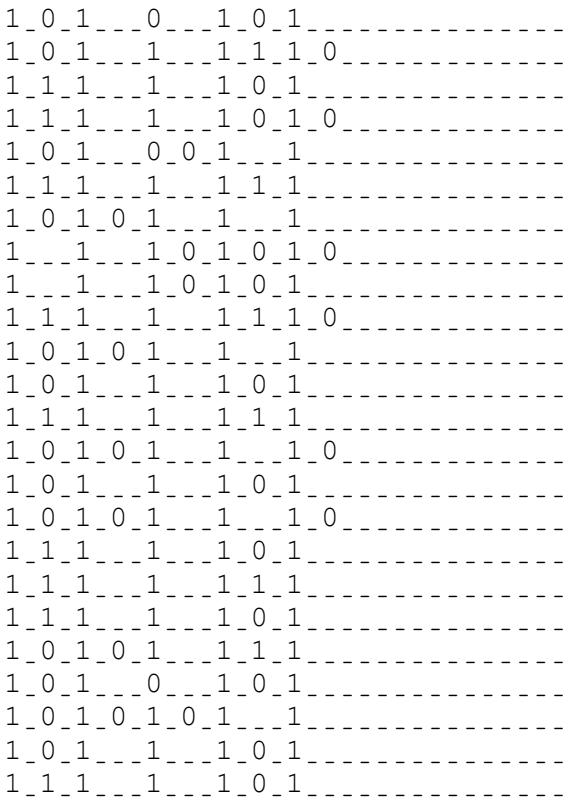
Till warm weather comes again
Then once more I see you peeping
Round about the tall tree roots
Nibbling at their fallen fruits

Fieldmouse fieldmouse do not go
Where the farmer stacks his treasure
Find the nut that falls below
Eat the acorn at your pleasure

But you must not steal the grain
He has stacked with so much pain
Make your hole where mosses spring
Underneath the tall oak's shadow

Pretty quiet harmless thing
Play about the sunny meadow
Keep away from corn and house
None will harm you little mouse

- **Stress markers metric distribution output**



- **Phones Assignment to the markers**

W|EH|R._DH|AH._EY._K|AO|R|N._T|AH|M._B|AH|L|Z._D|AW|N._____

W|EH|R._DH|AH._AE|SH._T|R|IY._SH|EH|D|Z._IH|T|S._B|EH._R|IY._____

W|IH|DH._Y|AO|R._F|ER._S|OW._S|AA|F|T._AH|N|D._B|R|AW|N._____

W|IH|DH._Y|AO|R._AY._S|OW._R|AW|N|D._AH|N|D._M|EH._R|IY._____

S|K|EH|R|S._L|IY._M|UW._V|IH|NG._DH|AH._____L|AO|NG._____

F|IY|L|D._M|AW|S._AY._K|AE|N._S|IY._Y|UW._P|AE|S._____

L|IH._T|AH|L._TH|IH|NG._IH|N._W|AH|T._____D|AA|R|K._D|EH|N._____

L|AY._____Y|UW._AO|L._DH|AH._W|IH|N._T|ER._S|L|IY._P|IH|NG._____

T|IH|L._____W|AO|R|M._W|EH._DH|ER._K|AH|M|Z._AH._G|EH|N._____

DH|EH|N._W|AH|N|S._M|AO|R._AY._S|IY._Y|UW._P|IY._P|IH|NG.____

R|AW|N|D._AH._B|AW|T._DH|AH._T|AO|L._____T|R|IY._R|UW|T|S._____

N|IH._B|AH._L|IH|NG._____AE|T._DH|EH|R._F|AA._L|AH|N._____

F|IY|L|D._M|AW|S._F|IY|L|D._M|AW|S._D|UW._N|AA|T._G|OW._____

W|EH|R._DH|AH._F|AA|R._M|ER._S|T|AE|K|S._____HH|IH|Z._T|R|EH._ZH|ER.__

F|AY|N|D._DH|AH._N|AH|T._DH|AE|T._F|AO|L|Z._B|IH._L|OW._____

IY|T._DH|AH._EY._K|AO|R|N._AE|T._____Y|AO|R._P|L|EH._ZH|ER._____

B|AH|T._Y|UW._M|AH|S|T._N|AA|T._S|T|IY|L._DH|AH._G|R|EY|N._____

HH|IY._HH|AE|Z._S|T|AE|K|T._W|IH|DH._S|OW._M|AH|CH._P|EY|N._____

M|EY|K._Y|AO|R._HH|OW|L._W|EH|R._M|AO._S|AH|Z._S|P|R|IH|NG._____

AH|N._D|ER._N|IY|TH._DH|AH._T|AO|L._OW|K|S._SH|AE._D|OW._____

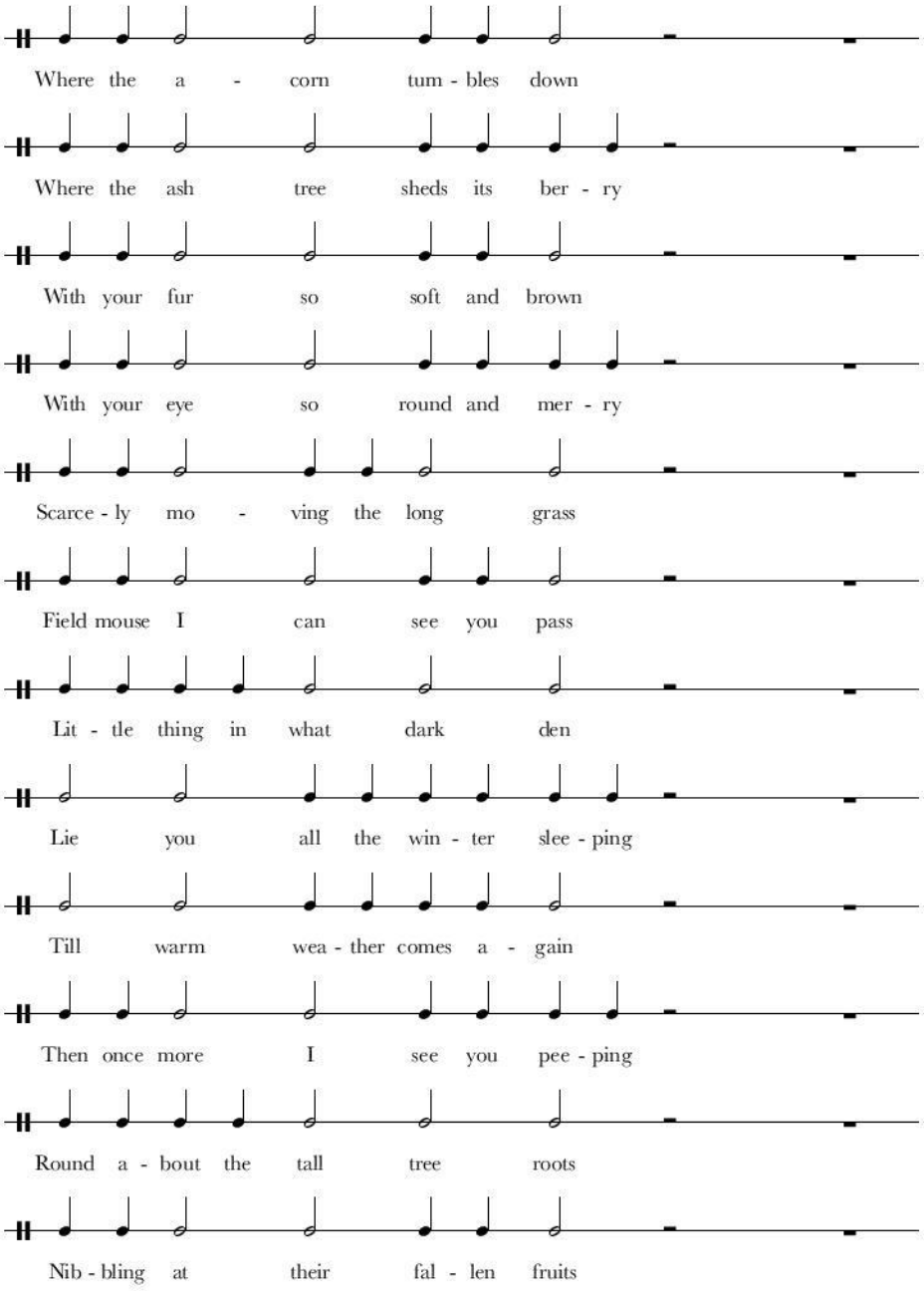
P|R|IH._T|IY._K|W|AY._AH|T._HH|AA|R|M._L|AH|S._TH|IH|NG._____

P|L|EY._AH._B|AW|T._DH|AH._S|AH._N|IY._M|EH._D|OW._____

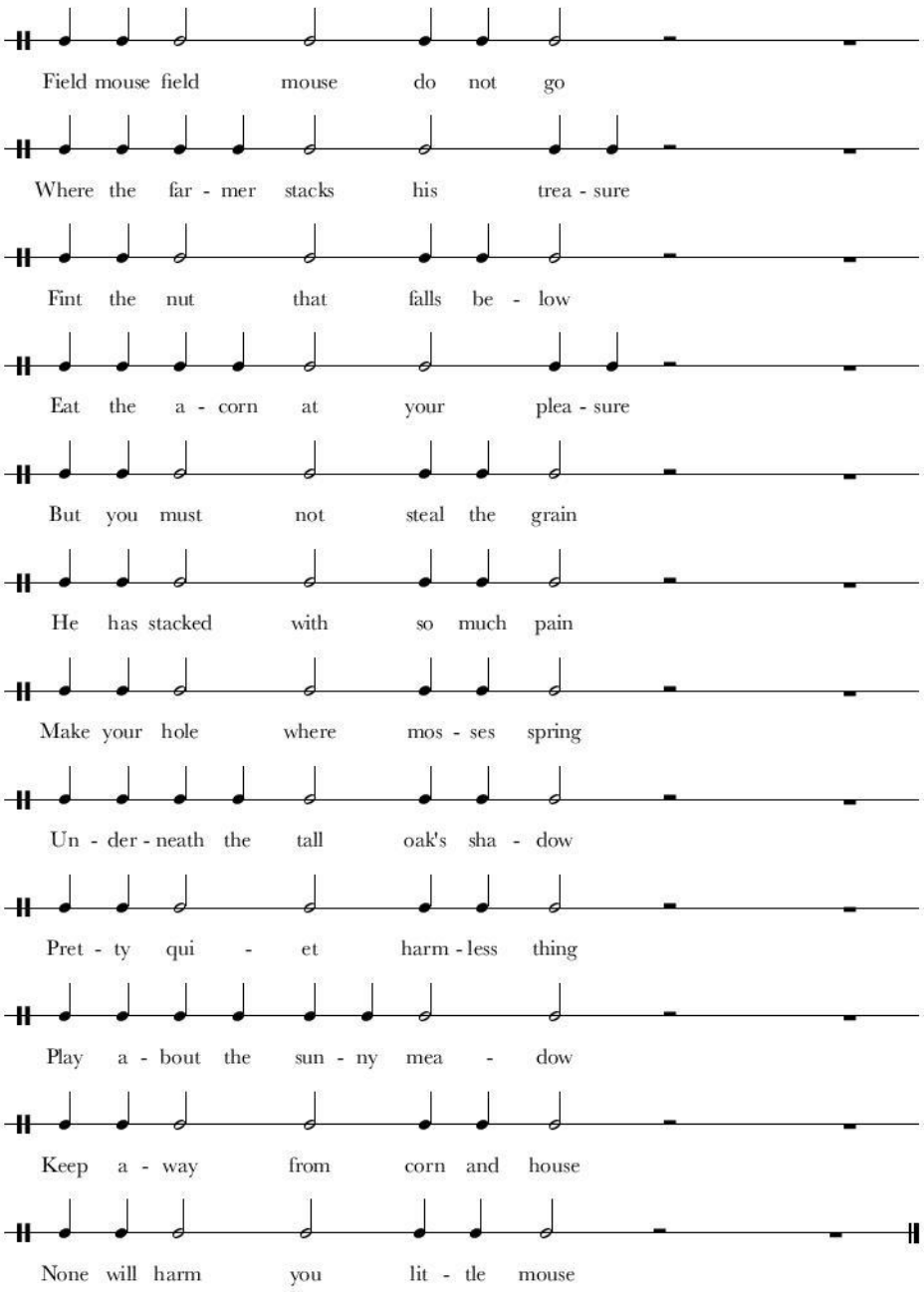
K|IY|P._AH._W|EY._F|R|AH|M._K|AO|R|N._AH|N|D._HH|AW|S._____

N|AH|N._W|IH|L._HH|AA|R|M._Y|UW._L|IH._T|AH|L._M|AW|S._____

- **Musical Notation Transcription**



2


Field mouse field mouse do not go
Where the far - mer stacks his trea - sure
Fint the nut that falls be - low
Eat the a - corn at your plea - sure
But you must not steal the grain
He has stacked with so much pain
Make your hole where mos - ses spring
Un - der - neath the tall oak's sha - dow
Pret - ty qui - et harm - less thing
Play a - bout the sun - ny mea - dow
Keep a - way from corn and house
None will harm you lit - tle mouse

5.2.4 The Lamb

- **Lyrics**

```
Little lamb who made thee
Dost thou know who made thee
Gave thee life and bid thee feed
By the stream and over the mead

Gave thee clothing of delight
Softest clothing woolly bright
Gave thee such a tender voice
Making all the vales rejoice

Little lamb who made thee
Dost thou know who made thee
Little lamb I'll tell thee
Little lamb I'll tell thee

He is called by thy name
For He calls Himself a lamb
He is meek and He is mild
He became a little child

I a child and thou a lamb
We are called by His name
Little lamb God bless thee
Little lamb God bless thee
```

- **Stress markers metric distribution output**

```
1___0___1___1___1___1___________
1___1___1___1___1___1___________
1___1___1_0_1___1___1___________
1___0___1___0___1_0_1___________
1___1___1___0___1_0_1___________
1___0___1___0___1_0_1___________
1___1___1___0___1_0_1___________
1___0___1___0___1_0_1___________
1___0___1___1___1___1___________
1___1___1___1___1___1___________
1___0___1___1___1___1___________
1___0___1___1___1___1___________
1___1___1___1___1___1___________
1___1___1___0___1_0_1___________
1___1___1_0_1___1___1___________
1___0___1___0___1_0_1___________
1___0___1___0___1_0_1___________
1___1___1___1___1___1___________
1___0___1___1___1___1___________
1___0___1___1___1___1___________
```

- **Phones Assignment to the markers**

L|IH.___T|AH|L.___L|AE|M.___HH|UW.___M|EY|D.___DH|IY._____

D|AA|S|T.___DH|AW.___N|OW.___HH|UW.___M|EY|D.___DH|IY._____

G|EY|V.___DH|IY.___L|AY|F._AH|N|D._B|IH|D.___DH|IY.___F|IY|D._____

B|AY.___DH|AH.___S|T|R|IY|M.___AH|N|D.___OW._V|ER._DH|AH._____

G|EY|V._______DH|IY.___K|L|OW._DH|IH|NG._AH|V._D|IH._L|AY|T._____

S|AO|F.___T|AH|S|T.___K|L|OW.___DH|IH|NG.___W|UH._L|IY._B|R|AY|T._____

G|EY|V._______DH|IY.___S|AH|CH._AH._T|EH|N._D|ER._V|OY|S._____

M|EY.___K|IH|NG.___AO|L.___DH|AH.___V|EY|L|Z._R|IH._JH|OY|S._____

L|IH.___T|AH|L.___L|AE|M.___HH|UW.___M|EY|D.___DH|IY._____

D|AA|S|T.___DH|AW.___N|OW.___HH|UW.___M|EY|D.___DH|IY._____

L|IH.___T|AH|L.___L|AE|M.___AY|L.___T|EH|L.___DH|IY._____

L|IH.___T|AH|L.___L|AE|M.___AY|L.___T|EH|L.___DH|IY._____

HH|IY.___IH|Z.___K|AO|L|D.___B|AY.___DH|AY.___N|EY|M._____

F|AO|R._______HH|IY.___K|AO|L|Z._HH|IH|M._S|EH|L|F._AH._L|AE|M._____

HH|IY.___IH|Z.___M|IY|K._AH|N|D._HH|IY.___IH|Z.___M|AY|L|D._____

HH|IY.___B|IH.___K|EY|M.___AH.___L|IH._T|AH|L._CH|AY|L|D._____

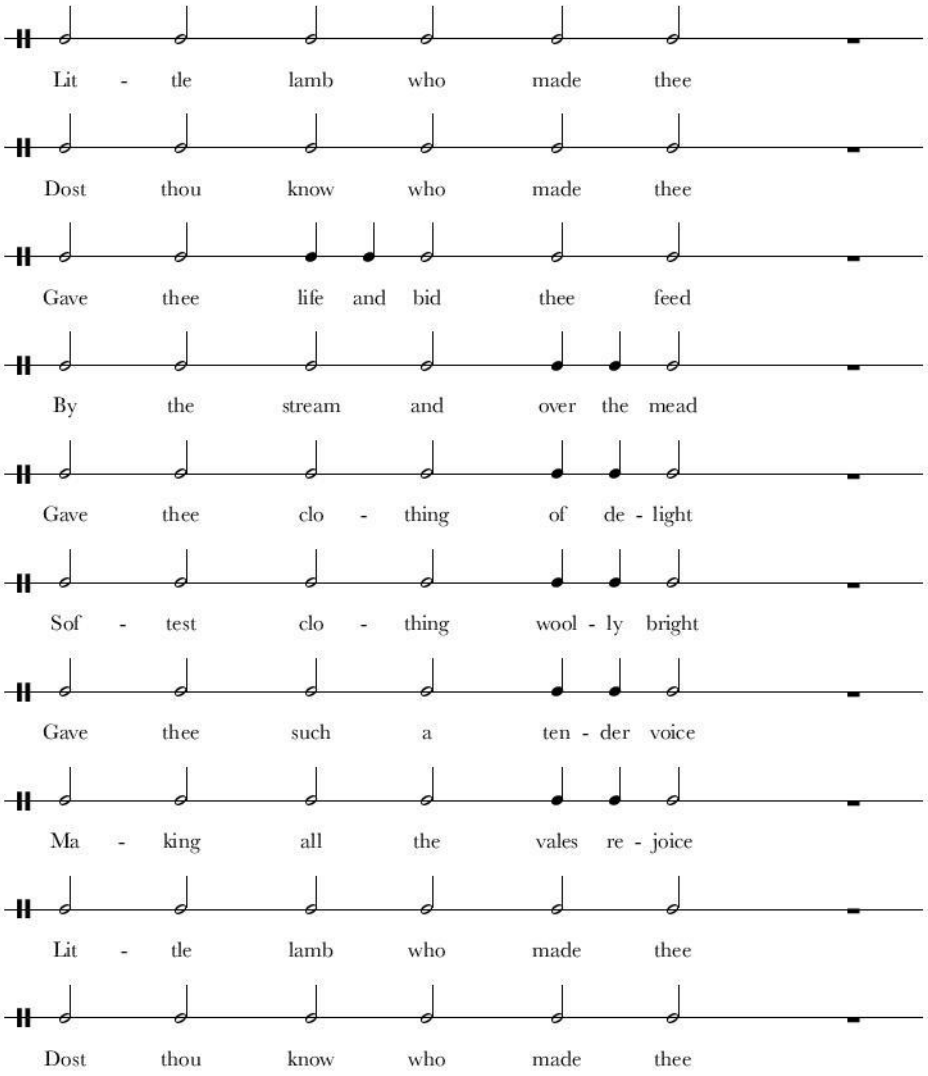
AY.___AH.___CH|AY|L|D.___AH|N|D.___DH|AW._AH._L|AE|M._____

W|IY.___AA|R.___K|AO|L|D.___B|AY.___HH|IH|Z.___N|EY|M._____

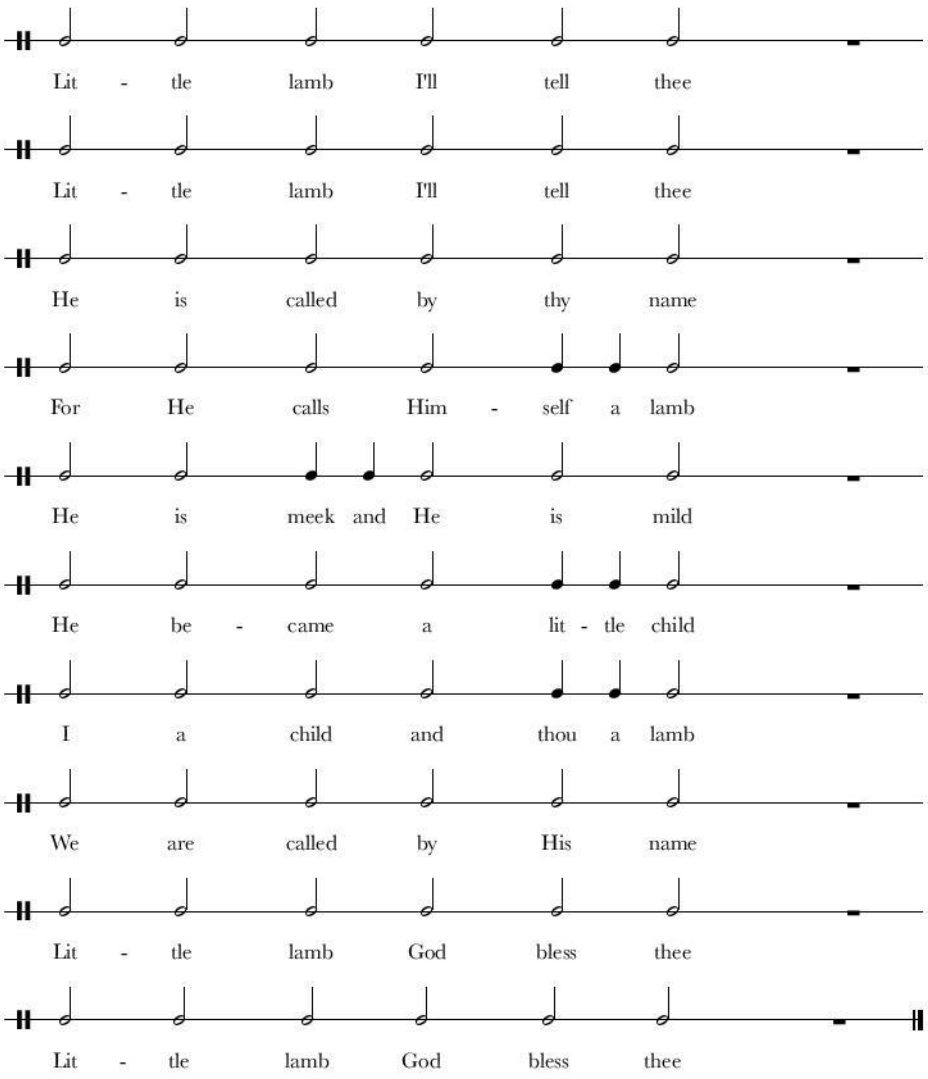
L|IH.___T|AH|L.___L|AE|M.___G|AA|D.___B|L|EH|S.___DH|IY._____

L|IH.___T|AH|L.___L|AE|M.___G|AA|D.___B|L|EH|S.___DH|IY._____

- **Musical Notation Transcription**



2


Lit - tle lamb I'll tell thee
Lit - tle lamb I'll tell thee
He is called by thy name
For He calls Him - self a lamb
He is meek and He is mild
He be - came a lit - tle child
I a child and thou a lamb
We are called by His name
Lit - tle lamb God bless thee
Lit - tle lamb God bless thee

6 Conclusion

In this project, I developed a system capable of analyzing a given musical lyric and generate a consistent metric distribution for that same lyric. The main motivation was to achieve an important starting point for the development of a complete system of musical generation based only in lyrics and the results are encouraging. Through a focused and reliable review of the literature, the principles inherent to the metric and rhythmic dimensions of music were studied and absorbed, as well as the parameters that build the relationship between music and lyrics. This research has led me to consider the phonetic characteristics of the text, specifically the lexical accents of words, as the most important pillar of the bridge between these two universes.

Therefore, I decided to redirect the focus of the research to the practical component. How would it be possible to analyze any text and from this analysis be able to extract useful data for generating musical content? What kind of tools would be available for this and how much would they be credible? Since I had already determined Python as the programming language to be used, the best tool to conduct the analysis of the text proved to be NLTK, which was designed for Python and already contained in itself the CMUdict module, a dictionary used for the phonetics analysis. Several tests were performed with different musical lyrics, until reach the conclusion that the best type of musical lyrics to be used as a basis for the development of the system would be lullabies. From then on, I wrote the code by modules, facing and solving a challenge at a time.

In the next chapter, I will summarize every step of the work from the text input to the final output. I will also be pointing the main obstacles and limitations of the project, as well as the contribution that I believe have given to the subject and what remains to be done in the future.

6.1 Summary

The practical process of this project was guided by the following principles:

- Lyrics and music can be directly connected and when aligned correctly tend to be better received and remembered by the listener;
- In the field of literature, meter and rhythm of a text are defined based on its phonetics features;
- The best musical metric structure for a given lyric is the one where all the verses fits the best.

Some rules were also created in order to respect both the concept of the system and its limitations:

- The text input will only be considered if written in English and if all its words are recognized by the system, so as not to compromise the final result;
- The lyrics format must be consistent and organized so that the system can find a good metric template for all of it. A musical lyric in which the difference of length from verse to verse is abrupt, or the length of all the verses is just too long or too short, will not be analyzed.

Once a lyric is received and accepted by the system, the operations which occur are the following:

1. All words are translated into a representation of their phonemes and its lexical stress markers are collected and stored in the order they appear on the verse.
2. Verses with an identical sequence of markers are removed and those who remain are defined as possible templates.
3. It's counted the number of strong stress markers from each of the templates, and its power of two is found and used to set the number of beats of the metric grid where it will be distributed.
4. All these templates are sent to a function that attempts to align in them all lexical stress markers from each of the verses in all possible ways. These possibilities will be classified and the template whose result of its intersection with every verse is the best, will be elected as the standard template for the lyric.

5. The phonemes of each verse are distributed according to this standard template and the results are returned by the system.

The main objective of the project was fully achieved: automatically generate a consistent metrical structure for a given lyric. However, the limitations are still quite a few, especially at the level of input. The system is not prepared to receive all kinds of lyrics, and at this stage only uniform texts are perfectly interpreted, as poems or children's songs. Also at the output level, there is a lot of work that can be done in order to make it more accessible to future projects.

6.2 Contribution of the Work

The contribution of my work is based on the breakdown of a first barrier to the generation of music content from a lyric. Since it assumes the text input as a unique source for this generation, it is necessary to extract as much useful information as possible from it, which in my experience the system is doing reasonably well, although it still has at its disposal an almost unlimited set of tools regarding to textual analysis, since only the CMUdict module was used from the NLTK's corpora, which contains hundreds of other modules. About this, I will be talking in the next chapter.

6.3 Future Work

As for future work, it is divided into three parts: the upgrade of the analysis that is already being done, the improvement of the output that is already being produced and addition of user interactions.

6.3.1 The Analysis

The analysis, although it’s producing the expected results at this stage, has a lot to improve. Not only by adding functions, but also by improving existing to reduce the system's limitations. I will mention below some of these enhancements:

- **Structure Analysis:**

  Recognizing structures and patterns within each lyric, not only in order to perceive its full form (types of verses, bridges, chorus, etc ...), as well as to find verses that are connected to other verses and stanzas with other stanzas. For this, it would be necessary to create a pattern recognition algorithm (word patterns but also phonetic accents patterns) and also search for rhymes;

- **Rhymes:**

  Once converted the verses into the phonetic representation, if we execute an inverted analysis, from the end of each verse to the beginning, we can compare the last phonemes of each verse. Defining a set of basic rules of rhyme, and a score, one can find out the groups of verses that rhyme and define how well they do it between them;

- **Treatment of Unrecognized Words:**

  This is the most ambitious step of this part. The biggest limitation of the system is the failure to recognize a few words. If they are misspelled it is good that the analysis does not go ahead, but in other cases it is a pure limitation and ignore those words do not seem to be the best option for the reliability of the results. Therefore, the rules of the process of conversion of words into phonemes would have to be carefully studied so that the system could simulate a result identical to the dictionary results for each new word it would receive.

6.3.2 The Output

- **Syllabification:**

  Although this process may prove to be performed in the analysis, I am putting it here because it's in the output that its result can be more useful.

Although being currently able to understand and clearly indicate where in the metric grid each word analyzed will be placed, the system only can create a presentation of this same distribution using phonemes. This is because although it can treat the word in separate phonemes, it cannot connect each of them to the characters of the word (the syllable) they represent. Taking out a syllabification process it could designate each of the phonetic accents to their respective syllable, making the output much clearer;

- **Visual Presentation:**

  The visualization of the results by the user is something that can also be improved. Using some visual/musical programming software (such as Processing, Max / MSP, etc.) one could create a dynamic presentation of the results, in a dedicated environment and with the possibility of a user interaction. This idea of user interaction will be better explained in the next chapter.

6.3.3 User Interaction

- **Subjective Decisions:**

  This feature would allow to the user make some decisions. The system is designed to classify all the possibilities generated and always select those that have the best rating. But sometimes there may be more than a possibility with the same score, or scores with very little difference. With this subjective decision, I am giving the user the opportunity to also access these hypotheses in order to be able to choose the ones that benefit them the most;

- **Syncopation:**

  This idea of syncopation came from the work of George Sioros and I think it might be interesting to apply it to my work as well. The distribution generated by the system is too square and linear, and it

would be an advantage allow the user to apply some swing effect, again depending on his taste. For this, it would be defined a priori a set of limits and developed a practical user interface.

7 Bibliography

Chomsky, N., & Halle, M. (1968), The sound pattern of English. New York: Harper & Row.

Lerdahl, F., & Jackendoff, R. (1983). A Generative Theory of Tonal Music. Cambridge, MA: The MIT Press.

Palmer, C., 86 Kelly, M. H. (1992), Linguistic prosody and musical meter in song. Journal of Memory and Language, 31:525-542.

Dawe LA, Platt JR, Racine RJ (1993), Harmonic accents in inference of metrical structure and perception of rhythm patterns. Percept Psychophys, 54:794–807.

J. P. G. Mahedero, A. Martinez, P. Cano, M. Koppen-berger, and F. Gouyon (2005), Natural Language Processing of Lyrics, in Proceedings of the 13th Annual ACMInternational Conference on Multimedia - MULTIME-DIA '05, pp. 475–478

Nichols E., Morris, D., Basu, S., Christopher, S. (2009), Relationships between lyrics and melody in popular music, Proceedings of the 10'th International Conference on Music Information Retrieval (ISMIR)

Nichols, E. (2009), Lyric-Based Rhythm Suggestion, in Proceedings of the International Computer Music Conference (ICMC).

Steven Bird, Ewan Klein, and Edward Loper. 2009. Natural Language Processing with Python. O'Reilly Media.

Monteith, K., Martinez, T. R., and Ventura, D. (2010), Automatic generation of music for inducing emotive response, in Proceedings of the First International Conference on Computational Creativity, 140–149.

S. Fukayama, K. Nakatsuma, S. Sako, T. Nishimoto and S. Sagayama, (2010), Automatic Song Composition from the Lyrics Exploiting Prosody of Japanese Language. Proc. Int. Conf. on Sound and Music Computing.

Monteith, K., Francisco, V., Martinez T., Gervas P., and Ventura D. (2011), Automatic Generation of Emotionally-Targeted Soundtracks, in Proceedings of the 2nd International Conference on Computational Creativity, pp. 60–62, Mexico City, Mexico,

Monteith, K., Martinez, T., Ventura, D. (2012), Automatic Generation of Melodic Accompaniments for Lyrics, in Proceedings of the Third International Conference on Computational Creativity, Dublin, Ireland

Sioros, G. (2016), "Syncopation as Transformation". Doctoral Dissertation, Universidade do Porto, Faculdade de Engenharia

8 Appendix

8.1 Python Code

```
1
2
3   # Importing libraries and defing variables and lists.
4   import itertools
5   import nltk
6   from nltk.corpus import cmudict
7
8   prondict = cmudict.dict()
9   arpabet = nltk.corpus.cmudict.dict()
10  filename='Hush, Little Baby'
11  inputfile = open('letras/'+filename+'.txt')
12  output1 = open('results/' + filename + ' - ' + 'outputMetrics.txt', 'w')
13  output2 = open('results/' + filename + ' - ' + 'outputPhonemes.txt', 'w')
14  my_text = inputfile.readlines()
15  my_text = [w.lower() for w in my_text]
16  simbolo = '<'
17  punctuation = '?.!,'
18  numbers=[0,1,2,3,4,5,6,7,8,9]
19  va = 0
20
21  res=[]
22  accents=[]
23  anacrusis=[]
24  anamount=[]
25  firstAnalysis=[]
26  temporaryStress=[]
27  templatesMetrics=[]
28  templates2test=[]
29  finalAssigns=[]
30  finalMetrics=[]
31  verseStress=[]
32  finalPhonemes=[]
33  phonemes=[]
34  systemOutput=[]
35
36  # Parsing the text file and finding the words in the dictionary
37  for line in my_text:
38      resultLine = []
39      for word in line.split(  ):
40          if word[0] in simbolo:
41              resultLine.append(word)
42          else:
43              if word not in prondict:
44                  resultLine.append(word)
45                  print(word, ': ', 'word not recognized')
46              else:
47                  resultLine.append(arpabet[word][0])
48      firstAnalysis.append(resultLine)
49
50  # Extracting the stress markers
51  for line in firstAnalysis:
52      newLine = []
53      for word in line:
54          newWord = []
55          for phone in word:
56              for char in phone:
57                  if char.isdigit():
```

```
                    newWord.append(int(char))
            newLine.append(newWord)
        accents.append(newLine)

    # Getting the phonemes from the text.
    for line in my_text:
        resultLine = []
        for word in line.split():
            if word[0] in simbolo:
                resultLine.append(word)
            else:
                if word not in prondict:
                    resultLine.append(word)
                else:
                    resultLine.append(arpabet[word][0])
        finalPhonemes.append(resultLine)

    # clean the line, keep only the numbers
    for line in accents:
        newLine = []
        if len(line) > 1:
            for token in line:
                for i in token:
                    if i in numbers:
                        newLine.append(i)
                    else:
                        pass
            temporaryStress.append(newLine)
        else:
            pass

    # find the anacrusis and mark their verse number
    for i in range(len(temporaryStress)):
        if temporaryStress[i][0]==0:
            r=0
            for x in temporaryStress[i]:
                if x==1 or x==2:
                    break
                r=r+1
            anacrusis.append(i)
            anamount.append(r)
            verseStress.append(temporaryStress[i][r:])
        else:
            verseStress.append(temporaryStress[i])


    # Generate the templates. Duplicate verses will be removed.
    for i in verseStress:
      if i not in templates2test:
        templates2test.append(i)

    # Function to convert the templates to metrical grids, with rests
    def finalTemplate(x):
        xacent1 = (x.count(1) + x.count(2))
        p2 = 2**(xacent1-1).bit_length()
        newFormat = ['_'] * (p2*4)
        div=int(len(newFormat)/p2)
```

```
    place1=0
    for char in range(len(x)):
        if x[char]==1 or x[char]==2:
            newFormat[place1] = 1
            place1 = place1 + div
        elif x[char]==0:
            lista = x[:char]
            car = x[char-1]
            if car == 1 or car == 2:
                valor = (((lista.count(1)+lista.count(2))*4)-
(int(div/2)))
                newFormat[valor]=0
            else:
                valor = (((lista.count(1)+lista.count(2))*4)-
(int(div/4)))
                newFormat[valor]=0
    finalFormat=[]
    for char in newFormat:
        if char == 0 or char == 1:
            finalFormat.append(char)
        else:
            finalFormat.append('_')
    return finalFormat
for line in templates2test:
    templatesMetrics.append(finalTemplate(line))

# Recursive function. Tries to fit every verse in every template and
returns all the possibilities for each fitting. X is the template, W is
the verse.
def recursive(x,w):

    xstress = x.count(1)
    ystress = (w.count(1) + w.count(2))
    dif = ystress - xstress
    newFormat = ['_'] * len(x)
    div = int(len(newFormat) / 8)
    st = xstress
    first = 0
    format = []
    tries = []
    rest = []
    result = []
    possibilities = []

    # Specific operations for the cases when the verse has more primary
stresses (1) than the template. Tries to fit the remaining ones in all
the possible beats.
    def more(line):
        onset=int((len(format)/8)-2)
        possibilities=[]
        while onset<len(line):
            if line[onset]=='_':
                line[onset]=1
                possibilities.extend(line)
                line[onset]='_'
                onset=onset+div
            else:
```

```
                onset=onset+div
        results = [possibilities[x:x + len(line)] for x in range(0,
len(possibilities), len(line))]
        return results

    # Specific operations for the cases when the verse has less primary
stresses (1) than the template. Tries to fit the missing ones (rests) in
all the possible beats.
    def less(line):
        onset = int((len(format) / 8))
        possibilities = []
        while onset < (len(line) - div):
            if line[onset] == 1:
                line[onset] = 3
                possibilities.extend(line)
                line[onset] = 1
                onset = onset + div
            else:
                onset = onset + div
        results = [possibilities[x:x + len(line)] for x in range(0,
len(possibilities), len(line))]
        return results

    # Fill the possibilities generated from the previous functions (less
and more) with the respective zeros.
    def zeros(x,y):
        s1=[]
        s0=[]
        x1=[]
        for c in range(len(y)):
            if y[c]==1 or y[c]==2:
                s1.append(c)
            elif y[c]==0:
                s0.append(c)
        for n in range(len(x)):
            if x[n]==1:
                x1.append(n)
        if y[-1] == 0:
            n=0
            for c in reversed(y):
                if c==1 or c==2:
                    break
                n=n+1
            del s0[-n:]
        for i in s0:
            y1=len([c for c in s1 if c<i])
            spot=x1[y1-1:y1+1]
            midpoint=int((spot[0]+spot[1])/2)
            if x[midpoint]=='_' or x[midpoint]==3:
                x[midpoint]=0
            else:
                try:
                    if x[midpoint+2]=='_':
                        x[midpoint + 2] = 0
                except:
                    if x[midpoint+1]=='_':
                        x[midpoint+1]=0
```

```
                    else:
                        return 'does not fit'
        return x

    # find the difference of primary stress between the template and the
verse to decide which specific function (less or more) must be executed.
    if dif >= 0:
        while st > 0:
            newFormat[first] = 1
            first = first + div
            st = st - 1
        for char in newFormat:
            format.append(char)
        reducedformat = format[:((xstress - 1) * div) + 1]
        rest = format[((xstress - 1) * div) + 1:]
        n=0
        p=0
        for c in reversed(w):
            if c==1 or c==2:
                break
            n=n+1
        while n>0:
            sp=int(div / 2)
            rest[sp+p] = 0
            n=n-1
            p=p+div
        tries = [reducedformat]
        while dif > 0:
            tries = [more(s) for s in tries]
            tries = list(itertools.chain.from_iterable(tries))
            dif = dif - 1
    elif dif < 0:
        while st > 0:
            newFormat[first] = 1
            first = first + div
            st = st - 1
        for char in newFormat:
            format.append(char)
        reducedformat = format[:((xstress - 1) * div) + 1]
        rest = format[((xstress - 1) * div) + 1:]
        if w[-1]==0:
            rest[int(div/2)-1]=0
        tries = [reducedformat]
        while dif < 0:
            tries = [less(s) for s in tries]
            tries = list(itertools.chain.from_iterable(tries))
            dif = dif + 1

    for line in tries:
        result.append(zeros(line, w))

    # output the results from this function (recursive).
    for line in result:
        if line == 'does not fit':
           pass
        else:
            possibilities.append(line+rest)
```

```
    if not possibilities:
        return 'does not fit'
    else:
        return possibilities

# Scoring function. Compare the template and each possibility inside each
fitting and select the best.
def scoring(x,y):
    def comparison(a,b):
        listfinal = []
        listfinal2 = []
        if len(a) != len(b):
            div = ((max(len(a), len(b)) / min(len(a), len(b))) / 2)
            d = [None] * div
            s = '_'
            if a < b:
                for c in a:
                    listfinal.append(c)
                    for i in d:
                        listfinal.append(s)
            else:
                for c in b:
                    listfinal2.append(c)
                    for i in d:
                        listfinal2.append(s)
        else:
            for i in a:
                listfinal.append(i)
            for j in b:
                listfinal2.append(j)
        sc=0
        firstLevel=[]
        secondLevel=[]
        ma=len(listfinal)
        di=int(len(verseStress)/4)
        f=0
        s=int(0+(di/2))
        while f<ma:
            firstLevel.append(f)
            f=f+di
        while s<ma:
            secondLevel.append(s)
            s=s+int(di)

        for i in range(len(listfinal)):
            if listfinal[i]!= listfinal2[i]:
                if i in firstLevel:
                    sc=sc+100
                elif i in secondLevel:
                    sc=sc+50
                else:
                    sc=sc+25
        return sc

    results=[]
    if y!= 'does not fit':
        for line in y:
```

```
            results.append(comparison(x,line))
    return results

# Creating the multidimensional arrays to host the results.
i, j = len(templatesMetrics), len(verseStress)
resultsArray = [[None for x in range(j)] for y in range(i)]
score = [[None for a in range(j)] for b in range(i)]
maxim = [[None for c in range(j)] for d in range(i)]
index = [[None for e in range(j)] for f in range(i)]


# Triggering the recursive function.
for t in range(len(templatesMetrics)):
    for v in range(len(verseStress)):
        resultsArray[t][v]=recursive(templatesMetrics[t],verseStress[v])

# Triggering the scoring function.
for t in range(len(templatesMetrics)):
    for p in range(len(verseStress)):
        score[t][p]=scoring(templatesMetrics[t], resultsArray[t][p])

# Drastically increasing the score values for possibilities that didn't
fit in the template (not very professional, but pragmatic solution. This
possiblities shall not be selected).
for line in range(len(templatesMetrics)):
    for i in range(len(verseStress)):
        if score[line][i]:
            maxim[line][i]=min(score[line][i])
        else:
            score[line][i]=[100000]
            maxim[line][i]=100000

for line in range(len(templatesMetrics)):
    for i in range(len(verseStress)):
        index[line][i]=[a for a, x in enumerate(score[line][i]) if x ==
min(score[line][i])]

for t in range(len(templatesMetrics)):
    sco=0
    for v in range(len(verseStress)):
        sco=sco+maxim[t][v]
    res.append(sco)
    sco=0

# Selecting the best template by finding the lowest score.
best=[i for i, x in enumerate(res) if x == min(res)]
select=best[0]

# Getting the results from the selected template again.
for i in range(len(verseStress)):
     finalMetrics.append(resultsArray[select][i][index[select][i][0]])

for line in finalPhonemes:
    phonemesTemp=[]
    if len(line)>1:
        for i in line:
            if i == ',' or i == '.':
```

```
                pass
            else:
                phonemesTemp.append(i)
        phonemes.append(phonemesTemp)

# Simple function to check if phonemes are stressed.
def phon(x):
    for s in x:
        if s.isdigit():
            return 'yes'

# Joining the phonemes that belong together.
for line in phonemes:
    lineAssigns=[]
    for word in line:
        assigns=[]
        store=[]
        for i in range(len(word)):
            if phon(word[i])=='yes':
                if not store:
                    assigns.append(word[i])
                else:
                    conc=store[0]+'|'+word[i]
                    assigns.append(conc)
                    store=[]
            else:
                if not store:
                    store.append(word[i])
                else:
                    if len(assigns)<1:
                        store[0]+='|'+word[i]
                    else:
                        assigns[-1]+='|'+store[0]
                        store=[]
                        store.append(word[i])
        if store:
            assigns[-1]+='|'+store[0]
        lineAssigns.append(assigns)
    finalAssigns.append(lineAssigns)

# Assigning the phonemes to their respective place in the metrical grid.
def assign(x,y,w,r):
    output=[]
    newY=[]
    t=0
    if w==0:
        for i in y:
            for c in i:
                newY.append(c)
    else:
        for i in y[r:]:
            for c in i:
                newY.append(c)
    for char in range(len(x)):
        if x[char]=='_':
            output.append(x[char])
        elif x[char]==3:
```

```
            output.append('_')
        else:
            f=[]
            for c in newY[t]:
                if c.isdigit():
                    pass
                else:
                    f.append(c)
            output.append(''.join(str(elem) for elem in f))
            t=t+1
    return output

# Triggering the assign function.
for s in range(len(finalMetrics)):
    if s in anacrusis:
        systemOutput.append(assign(finalMetrics[s], finalAssigns[s], 1,
anamount[va]))
        va=va+1
    else:
        systemOutput.append(assign(finalMetrics[s], finalAssigns[s], 0,
0))

# Placing the anacrusis in the metrical grid
for i in range(len(anacrusis)):
    val=anacrusis[i]
    ind=anamount[i]
    cur=0
    sp=ind-1
    di=0
    while cur<ind:
        le=int((len(systemOutput[val-1])/(len(systemOutput)/2))+di)
        systemOutput[val-1][-le]=finalAssigns[val][sp]
        finalMetrics[val - 1][-le] = 0
        di=di+2
        cur=cur+1
        sp=sp-1

# Output the results to text files.
for line in systemOutput:
    print(' '.join(map(str, line)))
    output2.write("%s\n" % ' '.join(map(str, line)))
for line in finalMetrics:
    print(' '.join(map(str, line)))
    output1.write("%s\n" % ' '.join(map(str, line)))
```